Specifying Mobile Network using a *wp*-like Formal Approach

Awadhesh Kumar Singh[*] Umesh Ghanekar[†]
Anup Kumar Bandyopadhyay[‡]

Abstract

The paper aims at providing a formal system, motivated by Dijkstra's weakest precondition logic, for specifying mobile network. The paper shows how mobility can be specified using a state and transition based approach, which allows mobile hosts to be treated as nodes in a traditional statically structured distributed system. Another goal is to reason formally about the possible behaviors of a system consisting of mobile components. The handover procedure serves as an illustration for the notation. The contribution of the paper is the development of a style of modeling and reasoning about the temporal properties that allows for a straightforward and thorough analysis of mobile systems.

Keywords*: weakest precondition, mobile computing, specification, verification, safety, handover.*

1 Introduction

Formal methods are classically considered difficult and time consuming [1]. Despite the fact, formal methods have witnessed a growing interest in the area of specification, development, and verification of systems. They are increasingly used to increase confidence in the quality, reliability and design of systems or part thereof [2]. Hall in his Seven Myths [3] asserted that mathematics for specifications should be easy. Lamport [4] has stated that a specification for the required system must not only be easy to write and understand, they should also be easy to use for verifying the correctness. People find formal specifications difficult to read because of the large use of the symbols [5]. The notational difficulties are more in writing specifications, due to the need for great attention to detail and correct use of mathematical statements. The formal approaches to the design of systems rely on reasoning about properties of the system [6]. Thus specifications to support such an approach should provide enough scope for reasoning about properties. Hesselink [7] regards Hoare triples [8] as the most adequate way to specify the systems. He adds further, one can use Hoare logic to define derivability of Hoare triples, but weakest preconditions form a more convenient semantic formalism that is sufficiently close to Hoare triples. Moreover, in practice, program verification using the inference rules of Hoare logic can be complicated, because intermediate assertions are needed between the statements. Therefore, one uses verification rules based on weakest precondition

[*] Department of Computer Engineering, National Institute of Technology, Kurukshetra 136119, India
Email: aksinreck@rediffmail.com
[†] Department of Electronics and Communication Engineering, National Institute of Technology, Kurukshetra 136119 India,
Email: umesh_ghanekar@rediffmail.com
[‡] Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata 700032, India
Email: anupbandyopadhyay@hotmail.com

[9, 10]. However, there may exist some situations when *wp* is not computable. Then we provide some additional information in order to use *wp* in program verification.

Mobile computing has evolved as a new distributed computing paradigm in recent years. It happened due to developments in both hardware and software technology of mobile networks where physical devices migrate from one location to another [11]. Mobile systems allow the user to access resources only via message passing, hence they are examples of distributed system. The mobility poses new challenges [12] to the message passing mechanism. Therefore, the formal methods, developed for specification and verification of distributed systems, may not prove equally effective in formalizing mobile systems. The usual approach to specify such systems is to devise a new language or logic. A new language or logic means a new semantics, new proof rules, and new tools. It isn't a very comforting thought that, when faced with a new problem domain, one must redo everything [13]. Therefore, the proponents of formal methods either augment the existing ones to make them applicable in the mobile scenario or suggest some anchored version of it. It is the case here.

We inherit useful intuition from the weakest precondition (*wp*) logic [14, 15, 16], developed by Dijkstra, for two reasons. First, the *wp*-logic [14, 15, 16] is elegant and conceptually simple because it uses first order predicate logic, which is very easy to apply because its syntax and semantics are well defined. Secondly, through *wp*-logic one can derive cause from effect, that is – precondition from post conditions. Therefore, it is much more sound and preferable being goal oriented [17, 18]. However, in some cases when *wp* is not computable, we work with a useful precondition (with respect to a post condition) that is not the weakest precondition. For example, in case of loops, we use (i) an invariant that is true before and after each iteration and upon termination and (ii) a bound function, to show that the loop terminates and also, to give an upper bound on how many iterations can be executed before termination occurs.

Dijkstra developed it originally for the specification of sequential systems. This paper presents a formal approach, motivated by *wp*-logic, for specification of mobile network, which is obviously an example of non-sequential system. Although the approach was presented first, under the name Split Precondition Logic (SPL) in [19], to verify a distributed mutual exclusion algorithm where processes communicate through shared variables in a traditional statically structured environment. The question of mobility did not arise there. Therefore, in the previous work, it remained unconcluded that the SPL can handle mobility. The present article is to demonstrate the power of the SPL in respect of mobility. Hence, we propose to extend the proof technology into the realm of mobile network, where processes communicate through message passing and also the nodes may change their physical positions while communicating. Therefore, the article is not simply, a case study or, about re-presenting a known approach; it also aims to demonstrate that the SPL is suitable for both, modeling and reasoning about the mobile systems. The present exposition is the result of a careful re-evaluation of the implications of mobility on the SPL, a model originally intended for statically structured distributed systems. Our hope is that the approach used here could be applied fruitfully to the systems with a complexity similar to current communication protocols. We use the term "approach" to avoid suggesting that we have a well-developed methodology. However, the concepts presented here are not introduced casually. The handover procedure has been chosen for the illustration. The handover protocol is the example of an infinite state mobile system. Our approach could successfully handle it through the abstractions of inter-process communication.

The remainder of the paper is organized as follows. In section 2, we give the basic elements of the framework. Section 3 contains a brief overview of the handover protocol. In section 4, we present the complete specification of the mobility in our framework. Section 5 is a discussion about some important concepts related to our specification method. Section 6

includes formal verification of the safety and liveness properties. Section 7 presents concluding remarks.

2 The Split Precondition Logic (SPL)

Like every formal language, the split precondition logic also has a well-defined syntax and semantics. The description of the mathematical model is being given in the following paragraphs.

A set $P.X$ of states and a set $P.R$ of state transition rules define a process P. On the similar lines a set $S.P$ of processes interacting through message transactions define a system S. The expression $in(P.x)$ represents that a process P is in state $P.x$. The initial state of process, which is predefined, is denoted by the expression $initial(P.x_0)$. The collection of states of all the processes belonging to set $S.P$ is termed as state SX of the system S.

A state transition rule represents the movement of process from one state to other. In order to fire any transition rule $P.r$ and eventual establishment of post condition Q, there exists a corresponding weakest precondition $wp(P.r, Q)$. We postulate the value of wp for a specific post condition Q and for a specific transition $P.r$. Though the name, weakest precondition, sounds similar, but wp, in our case, is different with the wp used by Dijkstra [14, 15, 16]. He used it to represent a predicate transformer, i.e., a function that maps any predicate (that expresses a required post condition) to another predicate (that expresses the precondition for the statement to terminate in a state that satisfies the post condition). The wp, in Dijkstra's logic, is computed from the program text, hence, it represents a precondition which is always weakest. Here, in split precondition logic, unlike Dijkstra's wp-logic, the weakest preconditions are not computed from the program text, but rather they provide a new model of the program, which reflects the original semantics of the program. Since it depends on atomicity considerations, therefore, it represents a precondition that may not always be weakest, in true sense. However, whenever, the precondition would be weakest, it would be similar to Dijkstra's predicate transformer wp. If the system state satisfies $wp(P.r, Q)$ then execution of $P.r$ will eventually establish the post condition Q. However, this is not guaranteed unless $wp(P.r, Q)$ is true before the execution of $P.r$. The 'precondition' is 'weakest' in this sense only. In our logic, weakest precondition has been splitted into two entities.
(i) $wsp(P.r, Q)$, termed as weakest self precondition and is related to the process P itself.
(ii) $wcr(P.r, Q)$, termed as weakest co-operation requirements and includes the co-operation requirements from other processes.
Thus, the total weakest precondition will be given by
$wp(P.r, Q) = wsp(P.r, Q) \wedge wcr(P.r, Q)$
The above expression justifies the appropriateness of the name Split Precondition Logic. Since, the co-operation requirements have already been included in the wp in our approach, separate proof of co-operation, as required in [20], is not necessary here. Any transition rule $P.r$ is described jointly by weakest precondition $wp(P.r, Q)$ and post condition Q. This scheme is illustrated in the following example.

Example
A system S is described by the following informal specification.
(1) There are two processes P_1 and P_2 in S.
(2) A message transfer occurs from P_1 to P_2 when
(a) An input command with P_1 as source is executed in P_2.
(b) An output command with P_2 as destination is executed in P_1.
(3) A process must wait until the other process is ready for message transaction, in other words the input and output command must be synchronized.

2.1 Formal System Specification

2.1.1 States of process P_1

	STATE	*SEMANTICS*
1.	P_1 . *beo*	state of P_1 before execution of the output command
2.	P_1 . *rts*	state of P_1 when it is ready to send a message. This state occurs just after the execution of the output command
3.	P_1 . *sent*	state of P_1 describing the fact that the message transaction is over

2.1.2 States of process P_2

	STATE	*SEMANTICS*
1.	P_2 . *bei*	state of P_2 before execution of the input command
2.	P_2 . *rtr*	state of P_2 when it is ready to receive a message . This state occurs just after the execution of the input command
3.	P_2 . *rec*	state of P_2 describing the fact that the message transaction is over

With reference to above states we can represent the state transitions between both the processes by following figure 1. The dashed line shows the co-operation required from the other process to execute the specified transition rule. The same can be specified by the transition rules given below the diagram.

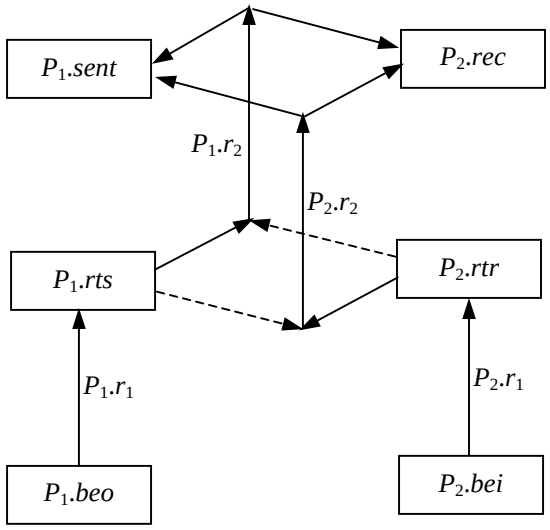

Figure 1. The State Transition Diagram

2.1.3 Transition Rules

Process P_1 ; identified by P_1 ;
States : P_1 . *beo*, P_1 . *rts*, P_1 . *sent*;
P_1 . r_1 : :
wsp(P_1 . r_1 , *in*(P_1 . *rts*)) = *in*(P_1 . *beo*)
wcr(P_1 . r_1 , *in*(P_1 . *rts*)) = *true*
end of P_1 . r_1;
P_1 . r_2 : :
def b = *in*(P_1 . *sent*) $\wedge$ *in*(P_2 . *rec*)
wsp(P_1 . r_2 , b) = *in*(P_1 . *rts*)
wcr(P_1 . r_2 , b) = *in*(P_2 . *rtr*)
end of P_1 . r_2;
end of transition rules;
Initial state : *in*(P_1 . *beo*)
end of process P_1.
Process P_2 ; identified by P_2 ;
States : P_2 . *bei*, P_2 . *rtr*, P_2 . *rec* ;
P_2 . r_1 : :
wsp(P_2 . r_1 , *in*(P_2 . *rtr*)) = *in*(P_2 . *bei*)

$wcr(P_2 . r_1 , in(P_2 . rtr)) = true$
end of $P_2 . r_1$;
$P_2 . r_2$: :
def b = $in(P_1 . sent) \wedge in(P_2 . rec)$
$wsp(P_2 . r_2 , b) = in(P_2 . rtr)$
$wcr(P_2 . r_2 , b) = in(P_1 . rts)$
end of $P_2 . r_2$;
end of transition rules.
Initial state : in($P_2 . bei$) ;
end of process P_2.

The transition rule $P_1 . r_2$ causes state transitions for both the processes P_1 and P_2. Hence, a corresponding transition rule, viz . , $P_2 . r_2$ is also defined in the process P_2. Though message transaction can occur between two processes only, we may have to include assertions about the states of more than one processes in $wcr(P . r , Q)$. This is necessary when P is allowed to accept a message from P_i only when P_j has not invoked a send command. Implementation of such a system is possible by using a control structure like pri-alt in occam.

2.2 Non-deterministic State Transition Rule

A non-deterministic state transition rule $P.r$ may include number of different sub-rules each of which requires a definite precondition to be satisfied for its execution. These preconditions will be called guards. Execution of a sub-rule will change the state of P as well as the state of one of the co-operating processes whose active co-operation is necessary for this execution. State transition in the co-operating process will be achieved by simultaneous execution of a state transition rule. If the preconditions for more than one sub-rule are satisfied then one of them is selected for execution. Selection procedure is non-deterministic and therefore, it is necessary to pass this information to the relevant co-operating process to produce the required state transition. The weakest precondition for a non-deterministic transition rule $P.r$ is obtained as follows.

Let there be n number of sub-rules denoted by $P.r^i : i = 1,\ldots,n$. On top of these sub-rules we assume a selector procedure, denoted by *select*, which makes the required non-deterministic selection .The post condition space for this procedure should therefore include a number of boolean variables denoted by

$s_i : i = 1,\ldots,m$. At each invocation the selector makes one such variable true. If a sub-rule $P.r^i$ has a post condition Q_i then

$s_i \Rightarrow wp(P.r^i, Q_i)$

Let B_i denotes the required guard for $P.r^i$, then the truth of this condition should ensure the selectability of s_i, i.e. ,

$B_i \Rightarrow wr(select, s_i)$

Where, $wr(select, s_i)$ is the weakest requirement that the procedure *select* may produce s_i. Using equations for s_i and B_i the rule $P.r$ is described as follows:

P.r *::*
$Q \equiv \exists\, i : Q_i$;
$wp(P.r, Q) = (\exists\, k : B_k) \wedge$
$(\forall\, i \bullet B_i \Rightarrow wr(select, s_i)) \wedge$
$(\forall\, j \bullet s_j \Rightarrow wp(P.r^j, Q_j))$;
end of $P.r$;

There are following two reasons for using *wr* in addition to *wp* here. We have used *wp* to specify the transition rules reflecting some state change(s) in the concerned process(es). The *select* procedure, which has been specified through the weakest requirement *wr*, does not reflect any state change. Rather, it reflects non-deterministic selection within a state transition rule. Secondly, it is a finer-grained specification. It gives more convenience to the system implementer.

2.3 The Property of a System

The operational model of a system can be specified by the state transition rules. These rules can be specified completely by their weakest precondition, post condition pairs. However, only operational specification may not be sufficient to specify the system requirements. In order to specify a system completely, along with the state transition rules the system properties must also be explicitly specified in a precise and implementation-independent manner. Various models allow the specification of the system properties in the form of invariants that should be preserved by all operations, pre- and post-conditions for operations or other forms of constraints on the system's behavior. Foe example, in our model, to specify a system invariant that must remain true before and after the execution of each state transition rule, there must exist a condition Q such that

$\forall\, i \bullet \forall\, m \bullet$

$\{wp(P_i \,.\, r_m\,,\, Q_{i\,,\,m}) \Rightarrow Q\} \wedge$

$(Q_{i\,,\,m} \Rightarrow Q)$

Similarly for a guarded command we have

$\forall\, i \bullet (B_i \Rightarrow Q) \wedge$

$(B_i \Rightarrow wp(P \,.\, r^i\,,\, Q_i)) \wedge (Q_i \Rightarrow Q)$

3 The Handover Protocol

Handover[1] is an important integral process of a modern day cellular system. It is the mechanism that transfers an ongoing call from one cell to another as a user moves through the coverage area of a cellular system. The switching should be performed in a fraction of a second and should be, generally, transparent to the two parties. In order to maintain the continuity and the quality of a call in progress, properly designed handover procedures are critical [21].

The decision to initiate the handover can be made by measuring several quantities such as signal levels the mobile station (MS) receives from the base stations (BS's), the distance from the BS's, and the bit error rate to estimate the quality levels of the concerned radio links. Signal level measurement is one of the commonly used criteria, especially in microcells where it could be the only reliable method available [22, 23].

A handover can be requested and/or forced by various points in the network. We can mainly distinguish between Mobile Services Switching Center (MSC) initiated handover (a network forced handover as a means of traffic load balancing) and mobile initiated/assisted handover. The Mobile Assisted Handover Algorithm (MAHO) was proposed, initially, by Agrawal and Holtzman [24]. The process is initiated only when the averaged signal strength received from the currently active BS drops below a minimum threshold. This situation is viewed as an indication that the mobile user is moving out of the initial cell into a neighboring cell. This parameter is determined by the network operator depending on the coverage by that particular BS. This parameter is a few dBs above a signal threshold for dropping. The difference between these two thresholds gives the mobile a chance to search for another BS before the call drops. The effective network is planned so that the mobile can find another BS whose received signal strength is above the minimum handoff threshold and not just above the threshold for dropping.

Once this need for a handover is detected, the mobile then scans the other BS (neighboring cells) and then hands over to another BS only if the signal strength of that BS is

[1] The terms "handover" and "handoff" are used interchangeably within the literature.

maximum amongst the rest of the base stations and exceeds the current one by *h* dB. *h* is the hysteresis level meant to avoid the repeated handoffs because of signal strength variations due to shadow fading. In a real system, the mobile continuously tracks the signals received from the neighboring base stations and sends the information to the BS. The BS keeps a track of these signal strengths and maintains a priority list of all the BS's according to the received signal level and uses the list to choose which BS to try for a handover when one is required. Similar to [25], the algorithm to decide whether to perform a handover or not is not specified in the current proposition. It is considered to be operator dependent. This algorithm is not investigated in our work, because our work starts when the decision has been made.

4 Specifying Mobility in the SPL

4.1 The Handover Procedure

Consider the following simplified handover procedure [26, 27, 28, 29] that involves all the steps required in a complete process. Let us have one Mobile Station (MS) and two Base Stations (BS), controlled by the same Mobile Switching Center (MSC). Each BS possesses one radio channel for communication with the MS. Each BS is linked to the MSC through a fixed channel, Assume the transfer of user data in one direction only from the MSC to the MS. Suppose that the MS is currently in the range of the first BS, as shown in Figure 2. We say that the BS is currently active, and that the other one is passive.

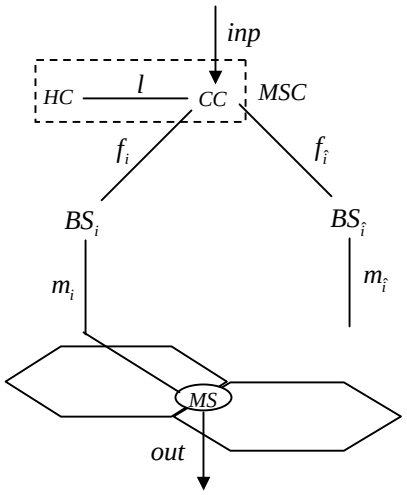

Figure 2. Initial Topology

Assume that the MSC consists of two parts, a Communication Controller (CC) and a Handover Controller (HC). The CC performs all communications over the fixed channels. The HC stores all unused radio channels (in our case just one) and may at any time tell the CC to initiate a handover. Let us take that it does it by transmitting on a private link the identity of the channel into which the handover should be made. In reality, the HC would be quite complex, containing many unused channels for many BS's, and it would be equipped to determine when and to which BS a handover should be made. Normally, the CC receives user data from the fixed network and sends them to the active BS. The BS sends them via the radio channel to the MS, which gives them to the user. The HC may eventually find out that the

handover in needed. It then changes its state from *idle* (idle) to *need handover* (need_ho). It then sends the information about the new radio channel, which should be used by the MS, to the CC via the private link. In this case, the CC sends a *handover command* (ho_cmd) message containing the information about the new channel to the active BS, which sends it to the MS via the old radio channel. Upon receipt of the handover command message at the MS, there are two possibilities as follows.

Ideally, the MS disconnects the old radio channel and establishes a connection with the new, currently passive BS by sending a *handover access* (ho_acc) message via the new radio channel to it, and continues the normal communication via the new BS. Upon receipt of the ho_acc message, the passive BS sends a handover complete (ho_com) message via the fixed link to the MSC and proceeds as active BS.

The other possibility is that the MS fails to establish a connection on the new radio channel and succeeds to reestablish the connection on the old channel. In this case, it sends a *handover failure* (ho_fail) message to the MSC via the old channel and BS, and resumes communication on the old channel as if no handover attempt had been made. After forwarding the ho_cmd to the MS, the active BS waits for response from either the MSC or MS. If it receives the ho_fail message, it forwards into the MSC via the fixed link and proceeds as active BS.

If the handover has succeeded, the other BS has become active and the MSC has been informed about it by ho_com. Upon receipt of the ho_com, the CC requests the information about now unused radio channel from the so far active BS with a *release channel* (relch) message. After its receipt, the BS sends the information to the CC and gets passive. When the CC receives the information, it forwards it through the private link to the HC, and, as it must have stopped to receive user data when the handover had started, it now resumes the normal operation in the mobile network with changed topology.

If the handover has failed, the CC gets the ho_fail from the MS via the old BS, and sends the information about the same radio channel, into which the handover should have been performed, back to HC. It then resumes the normal operation.

Whenever the HC requires a handover, it then waits for the CC to send it the information about which radio channel is not in use after the handover attempt. We have just told about when the CC sends it. Upon obtaining the information, it may require a handover into the unused radio channel.

Similar to [26, 27, 28, 29], the communication between the components of the procedure is assumed synchronous, that is, a process may send a message to another one only if it is ready to receive it. The sending and the receiving actions are then executed simultaneously and both processes pass to a new state. It is important to introduce synchronous communication, otherwise a deadlock could occur.

4.2 Handover in the SPL

4.2.1 Specification of Mobility

Now, we formalize the above-described handover procedure using split precondition logic. The SPL specifies the complete configuration of the system in terms of its components and its interactions. Also, any interaction between components is made explicit. Similar to other specification techniques, we also make certain assumptions, which provide framework for analysis of the procedure. Several CPUs may be present but memory hardware prevents simultaneous access to the same memory location. We also make no assumption about order of interleaved execution. Thus, we would be able to use assertional approach that is more convenient for formal verification [30]. Although we talk about a model of concurrency, we are actually modeling concurrency by a non-deterministic interleaving of atomic operations from the various processes. With an appropriate choice of atomic operations, almost any

concurrent system can be accurately modeled this way, in the sense that any safety or liveness properties proved about the model will be true of the system [31]. For instance, we can allow processes that share variables, or forbid it and request that processes send and receive messages instead. In both cases, the nature of composition is similar, that is, fair interleaving of atomic transitions [32].

The handover protocol, being an example of a network of processes that communicate by exchanging messages, can be modeled by using a shared variable to represent the communication channel between transmitting and receiving processes [26, 27, 28, 29]. With this structure it is easy to model a variety of assumptions about message transmission – for example, that the process delivers all messages safely or that it may non-deterministically lose or modify some of them. However, the similar approach has been used by Minsky et al. in [33] where a pair of agents, in heterogeneous distributed system, interact via tuple space to exchange messages between them and also by authors of [34] where mobile agents communicate with each other by communication space that is an implementation of tuple space. The tuple space is a logically shared memory because it provides the *appearance* of a shared memory but *does not require an underlying physical shared memory*.

The non-deterministic interleaving in our model of concurrency means that we make no assumption about the relative speeds of the processes and each process executes at non-zero speed. However, fairness implies that no processor is infinitely faster than another. This requirement is met, for example, by an implementation that provides a separate processor to execute each active process and fair scheduling of concurrent accesses to shared variables [31]. Therefore, we can formalize the handover procedure as a state transition system, consisting of following processes and their state transition rules, which will be able to include, and therefore manifest, all those execution sequences that could occur when the procedure is executed fairly.

It is assumed that the actions of different processes execute interleaved. We write the specification of the simplified handover procedure as composition of processes $CC, HC, BS_i, BS_{\hat{i}}$, and MS, representing both controllers, both Base Stations, and Mobile Station from the informal description, respectively. A channel (the 'frequency") for communication between two processes will be represented by a shared variable that can store only one message or have an 'empty' value. A process may write to a shared variable only if the variable is empty. The sending of a message will be represented by writing it to the variable with the 'empty' value. The receiving a message will be represented by reading the value of the variable when it is nonempty. Hence, the list of the shared variables is as follows.

inp – represents data input channel of the procedure

l – represents fixed channel for communication between CC and HC

f_i – represents fixed channel for communication between CC and BS_i

$f_{\hat{i}}$ – represents fixed channel for communication between CC and $BS_{\hat{i}}$

m_i – represents fixed channel for communication between MS and BS_i

$m_{\hat{i}}$ – represents fixed channel for communication between MS and $BS_{\hat{i}}$

out – represents fixed channel for delivering data to the user

We introduce two additional components, environment (E) that produces user data on inp and a user (U) that consumes the data on out. The channel names subscripted i and $\hat{i}$ represent the currently chosen channel and the unused channel to switch into through handover, respectively. Similarly, the BS names subscripted i and $\hat{i}$ represent the currently active BS and the passive BS that becomes active after the successful handover, respectively.

4.2.2 States of the Process *E*

	STATE	SEMANTICS
1.	$E.active_i$	Process E is active in channel i
2.	$E.passive$	Process E is passive

4.2.3 States of the Process *P*

($P \in \{HC, CC, BS_i, BS_{\hat{i}}, MS, U\}$ and $c \in \{inp, l, f_i, f_{\hat{i}}, m_i, m_{\hat{i}}, out\}$)

	STATE	SEMANTICS
1.	$P.idle$	Process P is idle
2.	$P.need\_ho$	Process P has found out that the handover is needed
3.	$P.ho\_cmd$	Process P has *handover command* massage
4.	$P.ho\_acc$	Process P has *handover access* message
5.	$P.ho\_com$	Process P has *handover complete* message
6.	$P.ho\_fail$	Process P has *handover failure* message
7.	$P.relch\_c$	Process P has *release channel c* message
8.	$P.free\_c$	Process P has *free channel c* message
9.	$P.$put$_c$	Process P has delivered message on channel *c*
10.	$P.rtr$	Process P is ready to receive message
11.	$P.$get$_c$	Process P has received message from channel *c*
12.	$P.$empty$_c$	Process P has removed message from channel *c*

The state transition rules are specified in appendix 1.

5 Discussion on the Approach

The basic approach is well introduced in the above section, but the formal details tend to obscure some important concepts. In this section, we attempt to explain these concepts without repeating the details of the underlying approach.

One may state that it must be possible for communication channel to lose messages. However, the specification does not require that the loss of messages be possible, since this would prohibit an implementation that guaranteed no messages were lost. The specification might require that some state transition may or may not occur despite the loss of messages. It does not require that all messages be delivered. Therefore, it need not be part of the actual specification [35].

In our approach, the behavior of a concurrent system has been specified as a sequence of transitions. One may interpret this situation as 'loss of concurrency'. However, this is not true. Generally, the temporal ordering among the discrete atomic transitions, describing the behavior of a system, is assumed a partial order. However, a partial order is equivalent to the set of all total orders that are consistent with it. Therefore, the extension, of partial order to a total order, converts the behavior of a system to be a sequence of state transitions. Moreover, there is no loss of information, since we are concerned with only safety and liveness properties. Therefore, concurrency remains in the form of non-determinism, that is, any two concurrent transitions are performed in either order.

From specifications, one should not be able to tell whether an operation is initiated by some software (subroutine call) or hardware action (raising a voltage on a wire). It must be specified in terms of implementation level concepts like subroutine names and voltages. The PUT-GET combination, used for communication between two processes through fixed

channel (that is shared variable), may be looked by someone as implementation level specification and it must be completely specified at the implementation level. Therefore, one may object that specifications are not implementation-independent. We agree that the implementer should have complete freedom in implementing the objects and operations that describe the internal behavior. Therefore, one must try to make specifications compact, by specifying in terms of the internal behavior that can be described with high-level concepts. Guttag and Horning [36] recognized the need of presence of two partitions in a specification, implementation-independent and implementation-dependent. If a process is executing PUT operation on the channel, the other process, which is trying to execute GET operation, on the same channel, should wait until the value is placed on the channel, otherwise it may get the null value. It is other way also. The process trying to execute PUT operation on the channel should wait until the other process has emptied the channel. Waiting is an implementation level concept that cannot be expressed in a general way, therefore, it is necessary, in the specification, to specify PUT and GET as atomic operations [37]. Also it does not mean that the entire PUT and GET operations have to be implemented as single atomic operation. It does mean that when the PUT operation is putting information on the channel, there must be some instant at which that information becomes visible to the process executing GET operation, and similarly some instant at which the GET operation finishes reading the information from the channel. If there is not such an instant, then the process executing the GET operation may obtain only a part of the information, since execution of the PUT operation is yet to finish [4]. Hence, the freedom of implementation is not restricted.

6 Verification of the Handover Protocol

The handover procedure is correct if it satisfies the following properties.

Safety

S_1: Each data item submitted from the environment (*E*) must be delivered to the user (*U*) only once.

S_2: There must exist linear precedence relation (represented as $d_i < d_j$) between the data items – that is, if the data item d_i has been produced, from *E* at *inp*, earlier than the data item d_j, then the relation must hold while delivering them to U.

Liveness

L_1: Each data item, produced from *E* to the network, will eventually be delivered to *U*.

L_2: *E* must produce data on *inp* infinitely often.

6.1 Proof

The handover procedure is a data-independent system – that is, data are not manipulated. Hence, the execution of no transition rule is dependent on their values. The data is only written to and read from some variables. Therefore, if it can be proved that the handover procedure works as perfect buffer; it will suffice for the verification of properties [27, 38, 39]. It has been assumed that the communication is synchronous; hence it would be sufficient to prove safety properties with just two different data and liveness properties with just one datum [39].

6.1.1 Proof of Safety

S_1: Assume the datum *d* is transmitted from BS_i to *MS*. In the SPL, this condition would be specified as follows.

$$in(MS.rtr) \wedge in(BS_i.\text{put\_}m_i) \supset in(MS.\text{get\_}m_i) \wedge in(BS_i.\text{empty\_}f_i) \quad (1)$$

In our approach f_i and m_i are shared variables, being communication channels. Therefore, RHS, of the equation 1, would imply following assignment to shared variables.

RHS $\supset$ m_i := d $\wedge$ f_i := 0 (2)

The communication between the components of the procedure has been assumed synchronous; it is obvious from the equation 2 that both actions, the writing of the datum d on the variable m_i and its removal from the variable f_i, happened together. Thus, the datum d, having been removed from its immediate source f_i, can never be assigned again to its immediate destination m_i. This is analogous to ideal 'cut' and 'paste' mechanism. Hence the process BS_i, after transmitting the datum d to the receiver process MS, removes the datum d from f_i, the channel that had supplied the datum d to it. Therefore, the BS_i behaves as buffer between CC and MS. Since, the $BS_i - MS$ pair was arbitrarily chosen, similar will be the case with any pair of processes connected through a direct channel; subsequently, no process will receive any datum more than once. Thus, the user would receive no datum, produced by the environment at the network, more than once.

S_2: Let us assume the contrary. The linear precedence relation $d_i < d_j$, between the data items, produced from E at inp, d_i and d_j, has been violated while delivering them to U. The data item d_j, produced later from E at inp, has overtaken the data item d_i, produced earlier from E at inp. In order to satisfy this condition, the datum d_i and d_j must have been, at least once, at the same point. There are only two possibilities.

S_2^1: The datum d_i and d_j must have been, at least once, at the same node, say BS_i. In the SPL, being data independent approach, the transmission and reception, at BS_i, can be specified as following transitions.

$$in(BS_i.rtr) \wedge in(CC.\text{put}\_f_i) \supset in(BS_i.\text{get}\_f_i) \wedge in(CC.\text{empty}\_inp) \quad (3)$$

$$in(BS_i.\text{get}\_f_i) \wedge in(MS.\text{empty}\_m_i) \supset in(BS_i.\text{put}\_m_i) \wedge in(MS.rtr) \quad (4)$$

In order to achieve a system state in which the datum d_i and d_j are at BS_i, there must exist transition A, which contains $in(BS_i.\text{get}\_f_i)$, as one of the process states, in its precondition and post condition both. The SPL specification of such transition will be as follows.

$$\exists A \bullet wp(A, in(BS_i.\text{get}\_f_i) \wedge .....) = in(BS_i.\text{get}\_f_i) \wedge ..... \quad (5)$$

Looking at the equation 3 and 4, we observe that no such transition exists. However, there may be another way to have such transition. There may exist two concurrent transitions A_1 and A_2; A_1 may contain $in(BS_i.\text{get}\_f_i)$, as one of the process states, in its post condition and A_2 may contain $in(BS_i.\text{get}\_f_i)$, as one of the process states, in its precondition or vice versa. Using the equation 3, 4, and 5, the SPL specification will be as follows.

$$\exists A_1 \bullet wp(A_1, in(BS_i.\text{get}\_f_i) \wedge .....) = in(BS_i.rtr) \wedge ..... \quad (6)$$

$$\exists A_2 \bullet wp(A_2, in(BS_i.\text{put}\_m_i) \wedge .....) = in(BS_i.\text{get}\_f_i) \wedge ..... \quad (7)$$

The required weakest precondition, for concurrent execution of transitions A_1 and A_2, will be the following conjunction of weakest preconditions specified in the equation 6 and 7.

$$in(BS_i.rtr) \wedge in(BS_i.\text{get}\_f_i) \wedge ..... \quad (8)$$

The process BS_i cannot be in two states together. Therefore, the equation 8 specifies an infeasible system state and will be reduced to false. The datum d_i, d_j, and the node BS_i were arbitrarily chosen. Thus, no two data items can be together at any node.

S_2^2: The datum d_i and d_j must have been, at least once, at the same channel. This is also an impossible system state. Since every channel is a shared variable, no variable can assume two values, d_i and d_j, together. As the datum d_i and d_j were arbitrarily chosen, no two data items could be together at any channel.

6.1.2 Proof of Liveness

L_1: L_1 will be true always, if an input transition with a datum occurs on *inp*, then an output transition with the datum eventually occurs on *out* [27, 39]. In our approach, the non-deterministic transitions have been specified by rules with mutually exclusive guards; hence, their fairness condition is already satisfied. Therefore, if the weakest precondition of a rule is satisfied, then the rule would be executed "sometime". Let *INIT* be the predicate that represents the proper initial state – that is, *E* is active to produce data at the *inp* and the *inp* is empty.

$$INIT = in(E.active_i) \wedge in(CC.\text{empty}\_inp) \quad (9)$$

The *INIT* specifies the weakest precondition of a transition rule. Therefore, we can write following expression.

$$INIT \supset in(E.\text{put}\_inp) \wedge in(CC.rtr) \quad (10)$$

The equation 10 represents weakest precondition of a transition rule, which would be fired "sometime" establishing its post condition. Similarly, looking at the other transition rules, we observe that the post condition of every transition rule contains the precondition of a transition rule, which would subsequently be fired. Therefore, we can represent this fact as following expression.

$$INIT \supset ..... \supset in(U.\text{get}\_out) \quad (11)$$

The equation 11 is a predicate logic expression. It can be expressed as following temporal logic expression. The operators $\supset$, $\Box$, $\rightarrow$, and $\rightsquigarrow$ have their usual meanings "implication", "always", "eventually", and "leads to" respectively. The dual $\Diamond$ to the operator $\Box$ is defined by

$\Diamond A \equiv \neg\Box \neg A$. Since $\Box \neg A$ asserts that *A* will never become true, we can read $\Diamond$ as "not never".

$$INIT \supset\rightarrow in(U.\text{get}\_out) \quad (12)$$

Now, we would use the following temporal logic equalities given in [40–46].

$A \supset\rightarrow B \equiv A \rightsquigarrow B$ and $A \rightsquigarrow B \equiv \Box (A \Rightarrow \Diamond B)$

The equation 12 can be expressed as follows.

$$INIT \rightsquigarrow in(U.\text{get}\_out)) \quad (13)$$

$$\Box (INIT \Rightarrow \Diamond (in(U.\text{get}\_out)) \quad (14)$$

The equation 14 specifies that for any time at which a datum is produced at the network, the user then or at some later time would receive it.

L_2: L_2 will be true if an input transition with the datum infinitely often occurs on *inp* [27, 39]. It can be false only when (i) there is single user in the network or (ii) if there is more than one user then no user wants to talk; which are never the case. In all other situations *E* would be ready to execute input transition, on *inp*, again and again. Hence, infinitely often datum would be produced on the *inp*. This is the reason behind our assumption, in the beginning, about the presence of an active process *E* that is ready to produce datum at the network. It suffices for the proof of the property **L_2**.

6.1.3 The Scope for designing Proof Assistant

Although the strength of our modeling approach is simplicity, accuracy has not been compromised for the sake of simplicity. Nevertheless, our approach needs careful human effort. However, in our logic, the correctness is ensured by proving assertions, which are formulas in the predicate logic. These formulas must be embedded into the system during its design phase. Dijkstra [14] also mentioned this in connection with the loop invariants. The proof of these formulas requires standard predicate logic rules and also the transition rules of

the system in question. Since we propose to specify a system by its transition rules, this formalization is available to us. Therefore, it should be possible to develop a rule-based system to evaluate the correctness of the assertions. One can also think of a proof system that may be intelligent enough to consult with the user and update its rule base. However, in order to limit the present exposition, we did not try that.

7 Conclusion

In the split precondition approach one specifies the protocol by describing a set of states and all transitions between the states that are allowed to occur. One of the advantages of the approach is that specifications can be written in friendly, familiar notations such as state-transition diagrams. The style of specification appears to be classical; the assigned meaning is new. The co-operation requirements have already been included, in the *wp*, in our approach. Hence, the separate proof of co-operation, as required in [20], is not necessary here. Therefore, our proof is simple and preferable being conceptually straightforward [47].

The handover protocol is an infinite state mobile system. In spite of this, our approach could translate it into the finite state non-mobile abstractions of inter-process communication. In order to be able to verify safety and liveness properties we introduced two additional processes, environment (*E*) that produces data on the *inp* and a user (*U*) that consumes the data on the *out*. The protocol has been specified as a composition of processes. The goal of the work was to examine the suitability of the approach, which is a variant of *wp*-logic, for specifying mobility using state based, shared variable concept. The idea of using shared variable to specify communication channel between two processes is similar to [27] where authors have specified non-deterministic action by several rules with the same guard and all rules of the system have been assumed to be weakly fair. However, in our approach such type of assumption is not needed, because the non-deterministic actions have been specified by several rules with mutually exclusive guards.

Rather than giving an abstract discussion about the approach, we have chosen to manifest its strength through specification and verification of the simple handover protocol involving all the steps required in a complete process, because some times good examples are more instructive than formal theories. The next step could be the discussion, in more generality, on specification and verification of more realistic ones. We believe that the semantic and syntactic constructs of the SPL have sufficient expressive power to model modern-age complex communication protocols. Moreover, the SPL uses logical reasoning, which is still the only way to prove correctness in the case of larger systems [48].

References

[1] R. M. B. E. Morsi and J. F. Leathrum. A formal specification and verification of GSM using the OSI-RM formal object model. In *Proceedings of the Applied Telecommunication Symposium ATS'03*, 30 March – 03 April, Orlando, Florida, pages 1–6, 2003.

[2] L. Andriantsiferana, B. Ghribi and L. Logrippo. Prototyping and formal requirement validation of GPRS: a mobile data packet radio service for GSM. In *Dependable Computing for Critical Application: Proceedings of the IEEE Conference*, 6–8 January, pages 109–128, 1999.

[3] J. A. Hall, Seven myths of formal methods. *IEEE Software*, vol. 7, September, pages 11-19, 1990.

[4] L. Lamport, Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2): 190–222, 1983.
[5] K. Finney, Mathematical notation in formal specification – too difficult for the masses?'. *IEEE Transactions on Software Engineering*, 22(2): 158-159, 1996.
[6] G. J. Doherty, J. F. Campos and M. D. Harrison, Representational reasoning and verification. *Formal Aspects of Computing*, 12(4): 260-277, 2000.
[7] W. H. Hesselink, Predicate transformers for recursive procedures with local variables. *Formal Aspects of Computing*, 11(6): 616-636, 1999.
[8] Z. Chaochen, C. A. R. Hoare and A. P. Raven, A calculus of durations. *Information Processing Letters*, 40(5): 269-276, 1991.
[9] M. Kirchner. Program verification with the mathematical soft-ware system Theorema. PhD thesis, Research Institute of Symbolic Computation, Linz, Austria, 1999.
[10] K. L. Ildiko. Program verification using Hoare logic. In *Computer Aided Verification of Information Systems – A Practical Industry Oriented Approach: Proceedings of the Workshop CAVIS'03*, 12th of February, e-Austria Institute, Timisoara, Romania, 2003.
[11] G. Smith. A formal framework for modeling and analyzing mobile systems. In *Australasian Computer Science: Proceedings of the 27th ACM International Conference ACSC 2004*, 26: 193–202, 2004.
[12] T. Imielinski and B. R. Badrinath, Mobile wireless computing: challenges in data management. *Communications of the ACM*, 37(10): 18–28, 1994.
[13] L. Lamport. Verification and specification of concurrent programs. In: J. W. de Bakker, W. –P. de Roever and G. Rozenberg (eds.), *A Decade of Concurrency – Reflections and Perspectives: Proceedings of the REX Workshop*, June, The Netherlands, Springer-Verlag, LNCS 803, pages 347–374, 1993.
[14] E. W. Dijkstra. A Discipline of Programming. Prentice Hall, Englewood Cliffs, NJ, 1976.
[15] N. Popov. Verification using weakest precondition strategy. In *Computer Aided Verification of Information Systems – A Practical Industry Oriented Approach: Proceedings of the Workshop CAVIS'03*, 12th of February, e-Austria Institute, Timisoara, Romania, 2003.
[16] K. R. M. Leino. Efficient weakest preconditions. Technical Report MSR-TR-2004-34, April, Microsoft Research, 2004.
[17] M. Ben-Ari. Mathematical Logic for Computer Science. Prentice Hall, Englewood Cliffs, NJ, 1993.
[18] E. D'Hondt and P. Panangaden. Quantum weakest preconditions. In *Quantum Programming Languages: Proceedings of the 2nd International Workshop QPL2004*, 12–13 July, Turku, Finland, pages 75–90, 2004.
[19] A. K. Singh and A. K. Bandyopadhyay, Verifying mutual exclusion and liveness properties with split preconditions. *Journal of Computer Science and Technology*, To appear, 2004.
[20] K. M. Chandi and B. A. Sanders, Predicate transformers for reasoning about concurrent computation. *Science of Computer Programming*, 24(2): 129–148, 1995.
[21] G. P. Pollini, Trends in handover design. *IEEE Communications Magazine*, March, pages 82–90, 1996.
[22] R. Vijayan and J. M. Holtzman, A model for analyzing handoff algorithms. *IEEE Transactions on Vehicular Technology*, 42(3): 351–356, 1993.
[23] F. Graziosi, M. Pratesi, M. Ruggieri and F. Santucci, A multi-cell model of handover initiation in mobile cellular networks. *IEEE Transactions on Vehicular Technology*, 48(3): 802–814, 1999.
[24] S. Agarwal and J. M. Holtzman, Modelling and analysis of handoff algorithms in multi-cellular systems. In *Proceedings of the IEEE Vehicular Technology Conference*, pages 300–304, 1997.

[25] A. Aaroud, S. E. Labhalla and B. Bounabat. Design of GSM handover using MARDS model. In *Information Technology and Applications: Proceedings of the 2nd International Conference ICITA2004*, 9–11 January, Harbin, China, 2004.

[26] F. Orava and J. Parrow, An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4(6): 497–543, 1992.

[27] T. Kapus and Z. Brezocnik. Verification of a mobile network in a shared-variables model. *Presented at the Verifications in New Orientations'97 Workshop*, May, Val Bregaglia, Switzerland, 1997.

[28] T. Kapus and Z. Brezocnik. TLA-style specification of a mobile network. In *Proceedings of the 23rd EURUMICRO IEEE Conference EUROMICRO'97*, 1–4 September, pages 440–447, 1997.

[29] T. Kapus and Z. Brezocnik. Verification of a mobile network handover procedure using Murϕ. In *Proceedings of the 6th Electro technical and Computer Science Conference ERK'97*, Z. A. J. C. Baldomir (ed.), 25–27 September, Portoro~, Slovenia, pages A/83–86, 1997.

[30] W. H. Hesselink, An assertional criterion for atomicity. *Acta Informatica*, 38(5): 343–366, 2002.

[31] L. Lamport, Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4(3): 155–495, 1982.

[32] M. Charpentier. An approach to composition motivated by *wp*. In *Fundamental Approaches to Software Engineering: Proceedings of the 5th International Conference FASE'2002*, April, Springer-Verlag, LNCS 2306, pages 1–14, 2002.

[33] N. H. Minsky, Y. M. Minsky and V. Ungureanu. Making tuple spaces safe for heterogeneous distributed systems. In *Applied Computing – Special Track on Coordination Models, Languages and Applications: Proceedings of the ACM Symposium ACMSAC'2000*, 19–21 April, Como, Italy, pages 218–226, 2000.

[34] L. Chunlin and L. Layuan. An agent-based approach for grid computing. In *Parallel and Distributed Computing Applications and Technologies: Proceedings of the 4th International Conference PDCAT'2003*, 27–29 August, pages 608–611, 2003.

[35] L. Lamport, A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1): 32–45, 1989.

[36] J. V. Guttag and J. J. Horning. Formal specification as a design tool. In *Principles of Programming Languages: Proceedings of the ACM Symposium*, January, Las Vegas, pages 251–261, 1980.

[37] L. Lamport. What it means for a concurrent program to specify a specification: why no one has specified priority. In *Principles of Programming Languages: Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium*, January, New Orleans, pages 78–83, 1985.

[38] K. Sabnani, An algorithmic technique for protocol verification. *IEEE Transaction on Communication*, 36(8): 924–931, 1988.

[39] P. Wolper. Specifying interesting properties of programs in prepositional temporal logic. In *Principles of Programming Languages: Proceedings of the 13th ACM Symposium*, pages 184–193, 1986.

[40] L. Lamport. "Sometime" is sometimes "not never": On the temporal logic of programs. In *Principles of Programming Languages: Proceedings of the 7th Annual ACM Symposium*, 28–30 January, Las Vegas, pages 174–185, 1980.

[41] E. A. Emerson. Temporal and modal logic. In: J. v. Leeuwen (ed.), *Handbook of Theoretical Computer Science*, vol. B, chapter 16, Elsevier Science, pages 995–1072, 1990.

[42] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer–Verlag, New York, 1992.

[43] Z. Manna and A. Pnueli. Temporal Verification of Reactive Systems: Safety. Springer-Verlag, New York, 1995.
[44] E. M. Clarke, O. Grumberg and D. A. Peled. Model Checking. MIT Press, 1999.
[45] B. Bérard et al. Systems and Soft-ware Verification: Model-Checking Techniques and Tools. Springer–Verlag, New York, 2001.
[46] F. Laroussinie, N. Markey and P. Schnoebelen. Temporal logic with forgettable past. In *Logic in Computer Science*: *Proceedings of the 17th Annual IEEE Symposium LICS'2002*, 22–25 July, Copenhagen, Denmark, pages 383–392, 2002.
[47] W. H. Hesselink, An assertional proof for a construction of an atomic variable. *Formal Aspects of Computing*, 16(4): 343–366, 2004.
[48] H. Dierks, Comparing model checking and logical reasoning for real-time systems. *Formal Aspects of Computing*, 16(2): 104–120, 2004.

APPENDIX 1

Transition Rules

The transition rules have not been categorized, under the name of the processes, as in the section 2.1.3. They have been shown interleaved for better understanding of the process of execution. However, the names of the transition rules bear their process names hence it can be easily identified to which process a particular transition rule belongs. The processes have more than one transition rules; therefore, the transition rules have been numbered also.

% Start of the communication %

$CC.r1::$

$Q = in(CC.rtr) \wedge in(E.\text{put}\_inp)$

$wsp(CC.r1, Q) = in(CC.\text{empty}\_inp)$

$wcr(CC.r1, Q) = in(E.active_i)$

end of $CC.r1$;

$CC.r2::$

$Q = in(CC.\text{get}\_inp) \wedge in(E.active_i)$

$wsp(CC.r2, Q) = in(CC.rtr)$

$wcr(CC.r2, Q) = in(E.\text{put}\_inp)$

end of $CC.r2$;

$BS.r1::$

$Q = in(BS_i.rtr) \wedge in(CC.\text{put}\_f_i)$

$wsp(BS.r1, Q) = in(BS_i.\text{empty}\_f_i)$

$wcr(BS.r1, Q) = in(CC.\text{get}\_inp)$

end of $BS.r1$;

$BS.r2::$

$Q = in(BS_i.\text{get}\_f_i) \wedge in(CC.\text{empty}\_inp)$

$wsp(BS.r2, Q) = in(BS_i.rtr)$

$wcr(BS.r2, Q) = in(CC.\text{put}\_f_i)$

end of $BS.r2$;

$MS.r1::$

$Q = in(MS.rtr) \wedge in(BS_i.\text{put}\_m_i)$

$wsp(MS.r1, Q) = in(MS.\text{empty}\_m_i)$

$wcr(MS.r1, Q) = in(BS_i.\text{get}\_f_i)$

end of $MS.r1$;

$MS.r2::$

$Q = in(MS.\text{get}\_m_i) \wedge in(BS_i.\text{empty}\_f_i)$

$wsp(MS.r2, Q) = in(MS.rtr)$

$wcr(MS.r2, Q) = in(BS_i.\text{put}\_m_i)$

end of $MS.r2$;

$U.r1::$

$Q = in(U.rtr) \wedge in(MS.\text{put}\_out)$

$wsp(U.r1, Q) = in(U.\text{empty}\_out)$

$wcr(U.r1, Q) = in(MS.\text{get}\_m_i)$

end of $U.r1$;

$U.r2::$

$Q = in(U.\text{get}\_out) \wedge in(MS.\text{empty}\_m_i)$

$wsp(U.r2, Q) = in(U.rtr)$

$wcr(U.r2, Q) = in(MS.\text{put}\_out)$

end of $U.r2$;

$U.r3::$

$wsp(U.r3, in(U.\text{empty\_}out)) = in(U.\text{get\_}out)$

$wcr(U.r3, in(U.\text{empty\_}out)) = true$

end of $U.r3$;

% Start of the handover %

$HC.r1::$

$wsp(HC.r1, in(HC.need\_ho)) = in(HC.idle)$

$wcr(HC.r1, in(HC.need\_ho)) = true$

end of $HC.r1$;

$HC.r2::$

$Q = in(HC.ho\_cmd) \wedge in(E.passive_i)$

$wsp(HC.r2, Q) = in(HC.need\_ho)$

$wcr(HC.r2, Q) = in(E.active_i)$

end of $HC.r2$;

$CC.r3::$

$Q = in(CC.rtr) \wedge in(HC.\text{put\_}l)$

$wsp(CC.r3, Q) = in(CC.\text{empty\_}l)$

$wcr(CC.r3, Q) = in(HC.ho\_cmd)$

end of $CC.r3$;

$CC.r4::$

$Q = in(CC.ho\_cmd) \wedge in(HC.\text{empty\_}l)$

$wsp(CC.r4, Q) = in(CC.rtr)$

$wcr(CC.r4, Q) = in(HC.\text{put\_}l)$

end of $CC.r4$;

$BS.r3::$

$Q = in(BS_i.rtr) \wedge in(CC.\text{put\_}f_i)$

$wsp(BS.r3, Q) = in(BS_i.\text{empty\_}f_i)$

$wcr(BS.r3, Q) = in(CC.ho\_cmd)$

end of $BS.r3$;

$BS.r4::$

$Q = in(BS_i.ho\_cmd) \wedge in(CC.\text{empty\_}l)$

$wsp(BS.r4, Q) = in(BS_i.rtr)$

$wcr(BS.r4, Q) = in(CC.\text{put\_}f_i)$

end of $BS.r4$;

$MS.r3::$

$Q = in(MS.ho\_cmd) \wedge in(BS_i.\text{empty\_}f_i)$

$wsp(MS.r3, Q) = in(MS.\text{empty\_}m_i)$

$wcr(MS.r3, Q) = in(BS_i.ho\_cmd)$

end of $MS.r3$;

$MS.r4::$

$Q_1 = in(MS.\text{put\_}m_{\hat{i}}) \wedge in(BS_{\hat{i}}.active)$

$Q_2 = in(MS.\text{put\_}m_i) \wedge in(BS_i.rtr)$

$R = Q_1 \vee Q_2$

$B_1 = in(MS.ho\_cmd) \wedge in(BS_{\hat{i}}.passive)$

$B_2 = in(MS.ho\_cmd)$

$\quad \wedge in(BS_i.\text{empty\_}f_i) \wedge \neg in(BS_{\hat{i}}.passive)$

$wp(MS.r4, R) = (B_1 \vee B_1) \wedge$

$(B_1 \Rightarrow wr(select, in(MS.r4.s_1))) \wedge$

$(B_2 \Rightarrow wr(select, in(MS.r4.s_2))) \wedge$

$(in(MS.r4.s_1) \Rightarrow wp(MS.r4^1, Q_1))) \wedge$

$(in(MS.r4.s_2) \Rightarrow wp(MS.r4^2, Q_2)))$

end of $MS.r4$;

$BS.r5::$

$Q_1 = in(BS_{\hat{i}}.ho\_acc) \wedge in(MS.\text{empty\_}m_{\hat{i}})$

$Q_2 = in(BS_i.ho\_fail) \wedge in(MS.\text{empty\_}m_i)$

$R = Q_1 \vee Q_2$

$B_1 = in(BS_{\hat{i}}.active) \wedge in(MS.\text{put\_}m_{\hat{i}})$

$B_2 = in(BS_i.rtr) \wedge in(MS.\text{put\_}m_i)$

$wp(BS.r5, R) = (B_1 \vee B_1) \wedge$

$(B_1 \Rightarrow wr(select, in(BS.r5.s_1))) \wedge$

$(B_2 \Rightarrow wr(select, in(BS.r5.s_2))) \wedge$

$(in(BS.r5.s_1) \Rightarrow wp(BS.r5^1, Q_1))) \wedge$

$(in(BS.r5.s_2) \Rightarrow wp(BS.r5^2, Q_2)))$

end of $BS.r5$;

$CC.r5::$

$Q_1 = in(CC.rtr) \wedge in(BS_{\hat{i}}.\text{put\_}f_{\hat{i}})$

$Q_2 = in(CC.rtr) \wedge in(BS_i.\text{put\_}f_i)$

$R = Q_1 \vee Q_2$

$B_1 = in(CC.\text{empty\_}f_{\hat{i}}) \wedge in(BS_{\hat{i}}.ho\_acc)$

$B_2 = in(CC.\text{empty\_}f_i) \wedge in(BS_i.ho\_fail)$

$wp(CC.r5, R) = (B_1 \vee B_1) \wedge$

$(B_1 \Rightarrow wr(select, in(CC.r5.s_1))) \wedge$

$(B_2 \Rightarrow wr(select, in(CC.r5.s_2))) \wedge$

$(in(CC.r5.s_1) \Rightarrow wp(CC.r5^1, Q_1))) \wedge$

$(in(CC.r5.s_2) \Rightarrow wp(CC.r5^2, Q_2)))$

end of $CC.r5$;

$CC.r6$ ::

$Q_1 = in(CC.ho\_com) \wedge in(BS_{\hat{i}}.\text{empty}\_f_{\hat{i}})$

$Q_2 = in(CC.ho\_fail) \wedge in(BS_i.\text{empty}\_f_i)$

$R = Q_1 \vee Q_2$

$B_1 = in(CC.rtr) \wedge in(BS_{\hat{i}}.\text{put}\_f_{\hat{i}})$

$B_2 = in(CC.rtr) \wedge in(BS_i.\text{put}\_f_i)$

$wp(CC.r6, R) = (B_1 \vee B_1) \wedge$

$(B_1 \Rightarrow wr(select, in(CC.r6.s_1))) \wedge$

$(B_2 \Rightarrow wr(select, in(CC.r6.s_2))) \wedge$

$(in(CC.r6.s_1) \Rightarrow wp(CC.r6^1, Q_1))) \wedge$

$(in(CC.r6.s_2) \Rightarrow wp(CC.r6^2, Q_2)))$

end of $CC.r6$;

$HC.r3$ ::

$Q_1 = in(HC.ho\_com) \wedge in(CC.\text{empty}\_l)$

$Q_2 = in(HC.ho\_fail) \wedge in(CC.\text{empty}\_l)$

$R = Q_1 \vee Q_2$

$B_1 = in(HC.\text{empty}\_l) \wedge in(CC.ho\_com)$

$B_2 = in(HC.\text{empty}\_l) \wedge in(CC.ho\_fail)$

$wp(HC.r3, R) = (B_1 \vee B_1) \wedge$

$(B_1 \Rightarrow wr(select, in(HC.r3.s_1))) \wedge$

$(B_2 \Rightarrow wr(select, in(HC.r3.s_2))) \wedge$

$(in(HC.r3.s_1) \Rightarrow wp(HC.r3^1, Q_1))) \wedge$

$(in(HC.r3.s_2) \Rightarrow wp(HC.r3^2, Q_2)))$

end of $HC.r3$;

$HC.r4$ ::

$Q = in(HC.idle) \wedge in(E.active_i)$

$wsp(HC.r4, Q) = in(HC.ho\_fail)$

$wcr(HC.r4, Q) = in(E.passive_i)$

end of $HC.r4$;

$BS.r6$ ::

$Q = in(CC.rtr) \wedge in(HC.\text{put}\_l)$

$wsp(BS.r6, Q) = in(CC.\text{empty}\_l)$

$wcr(BS.r6, Q) = in(HC.ho\_com)$

end of $BS.r6$;

$BS.r7$ ::

$Q = in(CC.relch\_m_i) \wedge in(HC.\text{empty}\_l)$

$wsp(BS.r7, Q) = in(CC.rtr)$

$wcr(BS.r7, Q) = in(HC.\text{put}\_l)$

end of $BS.r7$;

$BS.r8$ ::

$Q = in(BS_i.rtr) \wedge in(CC.\text{put}\_f_i)$

$wsp(BS.r8, Q) = in(BS_i.\text{empty}\_f_i)$

$wcr(BS.r8, Q) = in(CC.relch\_m_i)$

end of $BS.r8$;

$BS.r9$ ::

$Q = in(BS_i.relch\_m_i) \wedge in(CC.\text{empty}\_f_i)$

$wsp(BS.r9, Q) = in(BS_i.rtr)$

$wcr(BS.r9, Q) = in(CC.\text{put}\_f_i)$

end of $BS.r9$;

$BS.r10$ ::

$wsp(BS.r10, in(BS_i.free\_m_i)) = in(BS_i.relch\_m_i)$

$wcr(BS.r10, in(BS_i.free\_m_i)) = true$

end of $BS.r10$;

$CC.r7$ ::

$Q = in(CC.rtr) \wedge in(BS_i.\text{put}\_f_i)$

$wsp(CC.r7, Q) = in(CC.\text{empty}\_f_i)$

$wcr(CC.r7, Q) = in(BS_i.free\_m_i)$

end of $CC.r7$;

$CC.r8$ ::

$Q = in(CC.free\_m_i) \wedge in(BS_i.passive)$

$wsp(CC.r8, Q) = in(CC.rtr)$

$wcr(CC.r8, Q) = in(BS_i.\text{put}\_f_i)$

end of $CC.r8$;

$HC.r5$ ::

$Q = in(HC.free\_m_i) \wedge in(CC.\text{empty}\_inp)$

$wsp(HC.r5, Q) = in(HC.\text{empty}\_l)$

$wcr(HC.r5, Q) = in(CC.free\_m_i)$

end of $HC.r5$;

$HC.r6$ ::

$Q = in(HC.idle) \wedge in(E.active_{\hat{i}})$

$wsp(HC.r6, Q) = in(HC.free\_m_i)$

$wcr(HC.r6, Q) = in(E.passive_i)$

end of $HC.r6$;

% End of the handover %