Integration of Group Membership with Clock Synchronization in Time-Triggered Architecture (TTA)

presented by: Cosmin Condea
supervised by: Eyad Alkassar

The FlexRay Communication Protocol

Prof. Dr. Wolfgang J. Paul

Universität des Saarlandes
SS, 2005

Contents

1 Introduction

Time Triggered Architecture (TTA) is a distributed computer architecture designed for highly-dependable systems. Two essential algorithms that lay within this protocol are the clock synchronization and group membership. For both algorithms the correctness proof has been carried out *in isolation*. Consequently, it is of utmost importance to inquire whether the results of the separate analyses hold for the integration of the two services within TTA.
An in-depth view on the formal model, make us notice that there exists a mutual dependency between these two algorithms. First of all, group membership is described withing the *untimed synchronous system model* where the notion of time is abstracted away. Processors are assumed to be synchronized and run in lock step. Clearly, the service provided by clock synchronization in reality is actually hidden into the model, thus simplifying the formal analysis. As a result, group membership depends on clock synchronization.
On the other, clock synchronization is dealt with on the *time-triggered system model*, which takes into account explicit time issues. Furthermore, in TTA, a processor receives information about the other processors' clock readings via the normal exchange of data messages. Since a processor *only* accepts messages from processors it considers non-faulty (belonging to the same membership set), evidently clock synchronization depends on group membership.
A simple combination of the correctness proofs is not feasible due to the fact that the levels of abstraction at which the services are modeled do not match. For this reason, one has to develop a novel method that allows the integration of the two properties. Most importantly, given that the untimed synchronous system model is an abstraction of the time triggered system model under the assumption that processors are synchronized within a small bound, one should concretely apply this relationship with respect to clock synchronization and group membership in order to show that they both hold on the time triggered system layer.

2 Outline of the Approach

The first step in order to achieve the correctness proof of the integrated services, namely the group membership and clock synchronization, is to split the analysis into a series of successive intervals corresponding to synchronization intervals. The same "granularity" of the analysis was used in the proof of the clock synchronization and it also kept here because the synchronization algorithm is only executed on synchronization intervals. As a side note, keep in mind that a synchronization interval consists of several slots. The ultimate goal is then to show by induction on the synchronization intervals i, that for all i, clock synchronization and group membership hold ***concomitantly***, as expressed by the following theorem:

Theorem 2.1. *(INTEGRATION)*
The clock synchronization property and the group membership property hold in all synchronization intervals i:

$$\forall i : cs\_prop(i) \wedge mem\_prop(i)$$

□

This is the theorem exactly as it is stated in the PhD thesis of H. Pfeifer, my main source information. However, we have already mentioned that both properties have to hold on the concrete level, that is on the time-triggered system level. Still, group membership was proved on the more abstract level, that of synchronous time system. Taking into account the above mentioned aspects and for reasons of *clarity of expression* the theorem would, in my opinion, better be expressed in a way that clearly points out that we refer to the "refined" membership property which holds on the time-triggered level. My suggestion is to introduce another predicate $mem\_prop_{\textbf{REFINED}}$ and state the theorem as below:

Theorem 2.2. *(INTEGRATION - my version)*
The clock synchronization property and the group membership property hold **on the time-triggered system level** in all synchronization intervals i:

$$\forall i : cs\_prop(i) \wedge mem\_prop_{\mathbf{REFINED}}(i)$$

$\square$

Note: I will continue the explanation of the approach using my additional predicate which, in my honest belief, is helpful in providing us with a better insight into the matter. So, from now on:

- mem_prop : Group Membership on the untimed synchronous system level
- $mem\_prop_{\mathbf{REFINED}}$: Group Membership on the timed triggered system level

The next step is crucial for the integration. It is concerned with the "mapping" of the membership proof from the synchronous system model *down to* the time triggered system model which is unerringly entailed due to the differences between the two abstraction levels. Formally, this mapping(refinement) can be expressed in the following form:

$$\forall i : mem\_prop(i) \wedge cs\_prop(i) \Rightarrow mem\_prop_{\mathbf{REFINED}}(i)$$

This relation, as it is written above, is expressed for synchronization intervals i. However, since membership is proved on a slot basis, the refinement relation will also be valid for all slots. The cs_prop conjunct is motivated by the fact that this "transformation" requires that *synchronized clocks are present.*
Now we can reason about the main theorem with respect to its two consisting parts.

Group Membership Part: is trivially true; applying the refinement, we obtain that the membership holds not only on each synchronization interval, but actually on every slot in the time-triggered model.

Clock Synchronization Part: the proof of the clock synchronization goes along the following formulas:

$$cs\_req(i) \wedge cs\_prop(i) \Rightarrow cs\_prop(i+1)$$

In words, if clock synchronization requirements and clock synchronization property hold in interval i, then the clock synchronization holds in the $(i+1)$th interval. The cs_req predicate was kept as a precondition that had to be *unavoidably satisfied* when clock synchronization was ***detachedly*** proved. However, to serve the scope of the integration, this predicate must be interpreted now and, furthermore, we shall assure that it directly relies on the availability of the membership service, as the following formula reflects it:

$$\forall i : mem\_prop_{\mathbf{REFINED}}(i) \Rightarrow cs\_req(i)$$

Additionally, we assume the clocks to be initially synchronized, that is:

$$cs\_prop(0)$$

Finally, the desired integration proof of the group membership and clock synchronization can be accomplished by an easy induction on the number os synchronization intervals using the formulas above.

2.1 Benefits of the Approach

As stressed earlier, the synchronization service and the membership service depend on each other. The key point in their integration proof is to break this apparent circular dependency which has been completely achieved in the outlined approach by introducing an abstraction in two flavors.
First, group membership abstracts from clock synchronization by lifting the analysis to a higher, more abstract system model - the untimed synchronous system model. At this level, processors operate in lock step, that is, are completely synchronous and events such as message passing or state transitions occur instantaneously. Hence, no aspects of the actual synchronization proof were needed since synchronization is already part of the model.
The other form of abstraction is used in the analysis of the clock synchronization service. The dependency on group membership is abstracted away by introducing an abstract precondition that must be satisfied in order to complete the proof. In our case this abstract precondition involves an assumption about the correct reception of messages - an issue that is directly linked to membership. The actual interpretation of what a correct message reception means was kept uninterpreted, that is abstract, thus avoiding the need to include any detail from the membership service into the analysis of synchronization.
With this organization of the various models the efforts to integrate the two services have to concentrate ***only*** on resolving the abstractions. This means, on one hand, to show that the proof of group membership is valid within the time-triggered and, on the other hand, to show that group membership provides an adequate interpretation of correct message reception such that presupposed hypotheses for clock synchronization are satisfied. This is an enormous advantage of the presented approach. Inherently, this proof method totally eliminates the need of performing double or parallel induction like a standard approach would surely necessitate.
Moreover, as a concluding remark regarding the benefits, it is naturally more intuitive when one proves an integration theorem of two or more properties to attempt to reuse the standalone proofs of properties. This way, the complexity of the proof is kept feasible for formal analysis by employing some kind of *modularization*.

3 Abstract Principle of Integration

To emphasize the outlined approach at a higher level, the author introduces a general abstract principle that allows to integrate two mutually dependent properties that separately hold on two different levels of abstraction - one *abstract* and one *concrete* - such that in the end it can be proved that they both hold on the same (concrete) abstraction level. Certain requirements have to be fulfilled by the two properties for the principle to be applicable. Since clock synchronization and group membership will meet all the requirements, as it will be seen later on, their integration proof is actually an instance of this abstract principle.
For now, let us define all the entities of this abstract principle:

- uninterpreted types A(abstract), C(concrete) and $State$
- "initial" elements $initA : A$ and $initC : C$
- "successor" functions $nextA : A \to A$ and $nextC : C \to C$. These functions map an element of A or C to its successor.
- refinement function $refine : A \to C$. This function is intended to reflect the relationship between index types A and C.
- state accessor functions $stateA : A \to State$ and $stateC : C \to State$. These functions are intended to create a ***sequence of states*** with the index being of type A or C, respectively.

- predicates *PropA* and *PropC* over type state expressing the properties to be integrated. *PropA* is supposed to be expressed on an *abstract* whereas *PropC* on a more *concrete* level.
- predicates *ReqA* and *ReqC* over type *State* expressing certain conditions necessary to complete the proofs of *PropA* and *PropC*, respectively.

We are interested in the scenario when *PropA* and *PropC* can be proven independently of each other at different levels of abstraction but their proofs rely on particular preconditions *ReqA* and *ReqC*, where the preconditions for *PropA* can be satisfied by *PropC* and vice-versa. As the next theorem expresses it, the aim is to finally prove that *PropA* and *PropC* hold *simultaneously* on the concrete level of C.

Theorem 3.1. *(Integration of Properties)*

$$\forall a : c = stateC(refine(a)) \Rightarrow PropA(c) \wedge PropC(c)$$

□

Given the way we express the theorem and the fact that we need to prove propositions over *state sequences*, we need an induction principle over A.

Requirement 1. *(Induction over A)*
For all predicates P over type A, the following induction principle holds:

$$P(initA) \wedge (\forall a : P(a)) \Rightarrow P(nextA(a))) \Rightarrow \forall a : P(a)$$

□

The initial element of A should be refined into the initial element of C and furthermore, $refine$ should commute with the successor function: refining the successor of an element a of A is the same as refining a and then applying the successor function in C.

Requirement 2. *(Refinement Function)*
The refine function *refine* must satisfy the following two conditions:

- refinement of initial elements:
$$refine(initA) = initC$$
- refinement of successors:
$$\forall a : refine(nextA(a)) = nextC(refine(a))$$

□

The following two requirements state that both *PropA* and *PropC* can be proven in an induction-like manner at their respective abstraction level.

Requirement 3. *(Proof of PropA)*
The proposition *PropA* can be proved on the abstract level in an induction-like manner:

- initial element:
$$PropA(stateA(initA))$$
- inductive step:
$$\forall a : PropA(stateA(a)) \wedge ReqA(stateA(a)) \Rightarrow PropA(stateA(nextA(a)))$$

□

Requirement 4. *(Proof of PropC)*
The proposition *PropC* can be proved on the concrete level in an induction-like manner:

- initial element:
$$PropC(stateC(initC))$$
- induction step:
$$\forall c : PropC(stateC(c)) \wedge ReqC(stateC(c)) \Rightarrow PropC(stateC(nextC(c)))$$

□

The next two requirements illustrate how the *ReqA* and *ReqC* should be provided. To demonstrate the mutual dependency between *PropA* and *PropC*, we assume that each *ReqA* and *ReqC* need some "input" from the other level of abstraction.

Requirement 5. *(PropA provides ReqC)*
Let $c = stateC(refine(a))$. The requirement *ReqC* holds for state c on the concrete level if the property *PropA* holds in that state.

$$PropA(c) \Rightarrow ReqC(c)$$

□

With regard to this requirement, be aware of the fact that possibly *PropC* might not be extensible on the abstract level. For example, clock synchronization does not make much sense in an untimed synchronous system model.

Requirement 6. *(PropC provides ReqA)*
Let $c = stateC(refine(a))$. The requirement *ReqA* holds for state on the abstract level if both the property *PropA* holds in that state and *PropC* holds in the corresponding concrete state.

$$PropA(stateA(a)) \wedge PropC(c) \Rightarrow ReqA(stateA(a))$$

□

This is one essential requirement with regard to the adopted integration principle, as it defines how property *PropA* can be refined onto the concrete level.

Requirement 7. *(Refinement of PropA)*
Let $c = stateC(refine(a))$. The property *PropA* holds for an abstract state can be refined onto the concrete level provided *PropC* holds for the corresponding concrete state.

$$PropA(stateA(a)) \wedge PropC(c) \Rightarrow PropA(c)$$

□

The last requirement allows to map *PropA* back to the abstract level.

Requirement 8. *(Abstraction of PropA)*
Let $c = stateC(refine(a))$. If property *PropA* holds in c, then it also holds in the corresponding abstract state.
$$PropA(c) \Rightarrow PropA(stateA(a))$$

□

Having presented the above requirements, the proof of the Theorem 3.1 can consequently be developed:

Proof. The proof follows an induction on abstract type a.

Base Case:

- First, we establish $PropC(stateC(refine(initA)))$ using Req. 4 in combination with Req. 2
- To prove $PropA(stateC(refine(initA)))$ we can apply Req. 7. This requires to show also that $PropA(stateA(initA))$ holds, which can be accomplished using Req. 3 as well as a proof of $PropC(stateC(refine(initA)))$, as above.

Induction Step: Let $c = stateC(refine(a))$. We assume the both $PropA$ and $PropC$ hold for c.

- First, we consider $PropC(stateC(refine(nextA(a))))$. Using Req. 2 we get $PropC(stateC(nextC(refine(a))))$ and apply Req. 4. This requires to show that $PropC(c)$ holds, which is done using the induction hypothesis, and a proof of $ReqC(c)$. The latter can be established using Req. 5 where the precondition $PropA(c)$ is given by the induction hypothesis.
- In order to prove $PropA(stateA(nextA(a)))$ we use the proof principle for A as given by Req. 3. This requires us to show two things. First we must prove that $PropA(stateA(a))$ holds, which can be done using the induction hypothesis together with Req. refr8. Second, we need a proof of $Req(stateA(a))$. The latter can be accomplished by applying Req. 6. In order to be able make use of the last mentioned requirement, we must show that its precondition are satisfied. The first, $PropC(c)$ is given by the induction hypothesis, whereas the second, $PropA(stateA(a))$ is proved as above.

□

4 Proof of Integrated Services

To apply the abstract principle to the concrete case of group membership and clock synchronization we must assign each entity from abstract the principle an element from our "real" scenario. This assignments are illustrated in Table 4. Then, the integration proof for clock synchronization and group membership can be inherited from the generic proof presented in the previous section, under the condition that all requirements are met by the synchronization and membership services.
Most of them are rather easy to show, such as the induction principle on natural numbers or the properties of the refinement function. Requirements 3 and 8 require us to prove the validity of group membership within the untimed synchronous system model, whereas requirement 4 requires to prove the validity of clock synchronization within the time-triggered system model. These are beyond the scope of my topic and their elaborate proofs can be found in Chapter 3, respectively Chapter 4 in the PhD thesis of H. Pfeifer.
The more interesting proofs with regard to integration aspects are those of Req. 5 and Req. 7. In the following, I will continue with a sketch the proof of Requirement 5, while Requirement 7 is outlined in the next section. The fifth requirement corresponds to proving that if the group membership property holds on the time-triggered level in a given slot, then the requirements for clock synchronization, as expressed by *cs_req* are fulfilled.
The property *cs_req* refers directly to a predicate *correct_msg* which in the separate proof of clock synchronization was left uninterpreted and assumed to be valid. However, now, as we speak about the group membership as well and, additionally, we aim at integrating it with clock synchronization, it is time that we provide an interpretation for what *correct_msg* means. Recall that processors encode their membership information in the normal data messages they send. We also assume there exists

Integration Principle Entities	Instantiation for G.M. and C.S.
types A, C and $State$	
$State$	***function*** mapping *processors* to the *union of state elements* used for group membership and clock synchronization
A	***natural numbers*** in order to count *slots* in the untimed synchronous system model
C	***the product of types*** $Slot$ and $CTime$, where the first element denotes the *slot number* and the second element is the *logical clock time.* This choice is due to the fact that in the time-triggered model we refer to states at certain clock times; more precisely, we want to refer to the state of a processor at the end of the communication phase.
"initial" elements	
$initA : A$	0 (zero)
$initC : C$	$(0, schedule(0) + cmp\_start(0))$ Note: $schedule(i)$ represents the start time of slot i, whereas $cmp\_start(i)$ denotes the start of the computation time in slot i, or equivalently, the end of the communication phase of that particular slot.
"successor" elements	
$nextA : A \to A$	$sl \mapsto sl + 1$ (simply by adding 1)
$nextA : C \to C$	$(sl, T) \mapsto (sl + 1, schedule(sl + 1) + cmp\_start(sl + 1))$
refinement function	
$refine : A \to C$	a slot number sl from the abstract level is refined on the concrete level to the expression $(sl, schedule(sl) + cmp\_start(sl))$.
state accessor functions	
$stateA : A \to State$	for the abstract level, $stateA$ is given by the state accessor function of the untimed synchronous system model, $state^{ss}$, where $stateA(sl)(p) = state^{ss}(sl, p)$, for any processor p
$stateC : C \to State$	for the concrete level, the corresponding function $state^{tt}$ of the time-triggered model is used for $stateC$, where $stateC((sl, T))(p) = state^{tt}(T, p)$, for any processor p
predicates $PropA$ and $PropC$	
$PropA$	Group Membership
$PropC$	Clock Synchronization
predicates $ReqA$ and $ReqC$	
$ReqA$	`T` (True): since the group membership property can be proved in the synchronous system model without referring to clock synchronization
$ReqC$	cs_req: the necessary requirements needed by the clock synchronization

Table 1: From the Abstract Principle to G.M. and C.S.

some form of message decoding (for instance, by performing CRC tests), formally represented by the *decode_msg* predicate which allows a processor to extract the membership information `mem`. Then, we can state the following definition which says that a message is considered correct by a (receiver) processor if its membership information equals the receiver's view.

Definition 4.1. *(Correct Message)*
Let s be the currect state of a processor, and m the message it has received in the current slot. Message m is said to be *correct w.r.t.* s, denoted $correct\_msg(s, m)$ if

$$\texttt{mem}(s) = decode\_mem(m)$$

□

Moreover, we must assume that a non-faulty broadcaster will correctly encode its membership information into its message.

Assumption 1. *(Non-faulty Sender Correct Message)*
Let p be the current broadcaster in slot t, and s its current state. If p is non-faulty, then it correctly encodes its membership information in the message m it sends.

$$sent(t, p) \wedge p \in \mathcal{NP}^t \Rightarrow \texttt{mem}(s) = decode\_mem(m)$$

□

We are now able to prove Requirement 5 as an outcome of the following 3 statements:

1. the *agreement* property of group membership provides the fact that all non-faulty processors have the same membership sets
2. if the current broadcaster is non-faulty, its membership set is correctly encoded in the sent message (by the above assumption)
3. the receiver is non-faulty

Consequently, all non-faulty receivers consider the message correct. This, of course, under the assumption that during the particular slot we are reasoning about, no major fault was encountered that prohibited communication at all.

5 Refinement of Untimed Synchronous System View

This section presents a proof of the refinement relationship that allows properties of a distributed algorithm that are modeled and verified at the synchronous system level to be mapped onto the time triggered level. Thus, the final requirement, Req. 7, from the abstract principle presented in the previous section is proved and hence, the proof of integrating clock synchronization with group membership is completed.
In order to be able to establish a relationship between the two abstraction levels it it necessary to prove that a specific algorithm has the same behavior both on the time-triggered system level and on the untimed synchronous system level. This is accomplished through a traditional simulation proof: we show that the time triggered algorithm simulates - or implements - the untimed synchronous system. We therefore compare the execution of the algorithm on both levels. More exactly, we pay attention to the state of the processors that run the algorithm and the messages that are sent or received by these processors. Whereas in the untimed synchronous system model all processors take their steps simultaneously and instantaneously, the time triggered model introduces some time intervals during which these steps occur. Since at this level, we can only determine the state of a

processor with respect to its local time, we must require there exists some sort of *synchronization* in the time triggered system model - obviously accomplished by clock synchronization algorithm.
The other aspect pertaining to these abstraction levels is with regard to the fault model they employ. In order to be able to carry over any fault tolerance behavior that is established for the untimed synchronous system down to the time-triggered system, we must ensure that the two system descriptions use *the same fault model.* For this reason, the timing faults which are characteristic only to the time triggered system model must manifest themselves as either send or receive faults in the untimed synchronous system model.
As mentioned earlier, the ultimate goal concerning the refinement is to prove that at all (real) times, the state of a processor in the untimed synchronous system is the same as the corresponding state in the time triggered system. Since there is *no perfect synchronization*, we need to determine an instant at which all processors on the time-triggered level are in the same round in order to find a corresponding system state at the synchronous level. We know a slot on the time triggered level is divided into a communication phase and a computation phase (exactly in this order). When an algorithm is executed on the time triggered level, the processors change their internal state some time during the computation phase while during the communication phase their states remain unchanged. Consequently, the appropriate instants where we can compare the two system levels - synchronous and time triggered - are those belonging to the communication phase of the processors. However, besides imperfect synchronization, we must also take into account that some processors might have faster clocks than others and thus they begin their computation phase earlier. The interval of interest is, hence, what could be called the *global communication phase* of the system, that is the real-time interval during which *all* non-faulty processors are in their communication phase. The next definition expresses this concept in a formal way.

Definition 5.1. *(Global Communication Phase)*
The global communication phase, denoted $CommPhase(sl)$ of a givel slot sl is defined as the intersection of the communication phases of all non-faulty processors:

$$t \in CommPhase(sl) \iff \forall p \in \mathcal{NP}^t : in\_comm\_phase(sl,p)(t)$$

□

Having introduced the above notion, we are able to state the refinement relationship. Keep in mind the main requirement for the refinement relationship to hold is that processors of the system are synchronized.

Theorem 5.1. *(Refinement Relationship)*
Let sl denote some slot, i the synchronization round to which sl belongs. Then for all processors p that are non-faulty during the ith synchronization round, the state of the processors in the untimed synchronous system equals the corresponding state in time-triggered system during the global communication phase of sl, provided clocks of the processors are synchronized:

$$cs\_prop^i \wedge t \in CommPhase(sl) \Rightarrow state^{ss}(sl,p) = state^{tt}(T,p)$$

where $T = LC^i_p(t)$ - the logical clock of p in synchronization interval i.

Proof. Because t belongs to the global communication phase of slot sl, the processor p is in the communication phase at time t and hence $T = LC^i_p(t)$ belongs to the communication phase of sl.

$$schedule(sl) \leq LC^p_i(t) < schedule(sl) + cmp\_start(sl)$$

As p is non-faulty, we can derive that p has not changed its state since the beginning of the current slot, and hence

$$state^{tt}(T,p) = state^{tt}(schedule(sl),p)$$

Consequently, it suffices to prove that the corresponding states for p coincide at the start of each slot. This is proved in the separate lemma below. □

The Lemma 5.1 needed in the proof of the main theorem states that for all processors, the state descriptions at the start of each slot are equivalent. Its proof follows an induction on the slot number and, additionally, it requires that Lemma 5.2 also holds. The latter lemma states that a message reaches a receiver in its communication phase. Since the combined proofs of the two lemmas take approximatively two pages, I restrained myself to only expressing them. For details, please consult the PhD thesis of H. Pfeifer p.174-176.

Lemma 5.1. Given the preconditions of Theorem 5.1, the corresponding states of the non-faulty processors in the untimed synchronous system and the time-triggered system at the beginning of each slot coincide:

$$cs\_prop^i \Rightarrow state^{ss}(sl, p) = state^{tt}(schedule(sl), p)$$

Lemma 5.2. Provided that the clocks are synchronized, a message reaches a non-faulty receiver p in its communication phase.

$$cs\_prop^i \Rightarrow in\_comm\_phase(sl, p)(arr\_time(sl, p))$$

5.1 Constraints on Timing Parameters

The precision δ of the clock synchronization algorithm imposes several constraints for the various timing-related entities involved, such as the length of the communication phase and computation phase of the broadcasting slots, respectively.

Constraint 1: The time at which the sender of a slot starts its broadcast must be at least δ time units into its communication phase in order to allow the processor with the slowest clock to reach the communication phase.

Constraint 2: The communication phase must be long enough to ensure that even the processor with the fastest clock has not already started its computation phase when the message of a slow processor reaches it.

6 Summary and Conclusions

In his work, the author presented a formal proof of the integration of group membership with clock synchronization. The apparent circular dependencies between the two algorithms have been separated by expressing the group membership protocol within a model that abstracts from clock synchronization, while the formalization of the latter is parameterized with abstract properties regarding group membership. The sound integration follows an inductive argument on the number of synchronization intervals. For the inductive step in this proof it is necessary to show that the abstractions can be resolved. On one hand, this means that the proof of group membership must be transferred from the synchronous level to the time triggered level. On the other hand, it must be shown that group membership provides a good interpretation of abstract entities in the synchronization model such that the presupposed hypotheses for clock synchronization are met. The overall integration proof was presented in detail, by showing that it can be obtained as an instantiation of a more general abstract principle.

References

[1] H. Pfeifer: Formal Analysis of Fault-Tolerant Algorithms in the Time-Triggered Architecture, PhD Thesis, Univ. Ulm, 2003

[2] J. Rushby: An Overview of Formal Verification For the Time-Triggered Architecture, September 2002, LNCS Vol. 2469, pp. 833105.

[3] H. Pfeifer: Formal Verification of the TTP Group Membership Algorithm, Proc. of FORTE/PSTV 2000. Kluwer Academic Publishers, pp. 3-18, October 2000.