ソフトウェア工学 104－5
（1995．5．31）

Methodology for the Semantical Behavior Checking of Distributed Systems Specification

Issam A. Hamid
Tohoku University of Art & Design
Yamagata, JAPAN

Reinhard Gotzhein
University of Kaiserslautern,
Kaiserslautern, Germany

Abstract

The task of modeling and specifying distributed systems can be subdivided into two major steps. Firstly, there is the system architecture addressing structural aspects. Secondly, there is the system behaviour defining the allowed sequences of visible actions performed by system components at external interfaces. In previous results it has shown how static system architectures, i.e., architectures that remain unchanged during system execution, can be modeled, specified, and refined using the formalism of first-order temporal logic. In this paper these results will be generalized to cover some aspects of dynamically evolving architectures of distributed systems. Implications on the notion of architectural refinement are addressed, too.

分散システム仕様を検査する意味論的振る舞い

アイサム　ハミド
東北芸術工科大学
情報デザイン学科、山形

レインハリド　ゴッザイン
カイザーローテルン大学
情報工学科，カイザーローテルン，ドイツ

分散システムをモデル化し仕様化するタスクは、２つの主要な段階に分けることができる。最初に、システムアーキテクチャは、構造的側面に位置づけられる。次に、システムの振る舞いは、システムコンポーネントを実行する可視的なアクションのシーケンスを定義する。過去の研究成果では、時相論理を形式化することにより、いかにして静的アーキテクチャがモデル化され、仕様化され、更新されるかを示した。本論では、分散システムのアーキテクチャを動的に進化させるいくつかの側面を汎化することを示す。

1 Introduction

Today, large systems are usually distributed and concurrent. However, distributed systems are very difficult to design and to implement, particularly because of the risk of timing-related errors. The rate of failure of distributed systems is many times higher than for sequential systems. Therefore, the topic of design methods and techniques for distributed systems is currently one of the most important research areas in Computer Science.

The task of modeling and specifying distributed systems can be subdivided into two major steps. Firstly, there is the system architecture addressing structural aspects. Secondly, there is the system behaviour defining the allowed sequences of visible actions performed by system components at external system interfaces. In [Got93], we have argued that specifying the system architecture should be among the first design steps, and be as rigorous and formal as the specification of the system behaviour. Therefore, we have proposed a formal framework capable of addressing both aspects. This is different from other techniques which do not give a semantics to the structure of a specification and thus to the architecture of the specified system. A restriction of the approach has been that only static system architectures, i.e. architectures that remain unchanged during system execution, can be modeled, specified, and refined. However, large systems are usually evolving dynamically, and therefore require the possibility of treating dynamical changes of their structure.

In the Basic Reference Model for Open Systems Interconnection ([ISO81]), e.g., the concept of service-access-point is introduced: "a service-access-point is the access means by which a pair of entities in adjacent layers use or provide services"; and: "an (N)-service-access-point is only attached to one (N+1)-entity at a time"; "an (N)-service-access- point may be reattached to the same or another (N+1)-entity". Clearly, this indicates that certain dynamical changes of the system architecture during execution are within the scope of the Basic Reference Model and should therefore be formally specified.

Another example where dynamic architectures are used is the concept of the federated trader stemming from the area of Open Distributed Processing (ODP, [ISO93]). By means of the trader, a service user can dynamically establish and release an association with a service provider. This means that some mechanism is required to temporarily attach a service user and a service provider to a common interaction point. Further examples subject to architectural evolution are mobile communication and Intelligent Networks.

In this paper, we extend and generalize previous results on modeling and specifying architectures of distributed systems by treating dynamic architectures. The organization of the paper is the following: in Section 2, we give an outline on how to model, specify and refine static system architectures. In Section 3, this model is extended and generalized; implications on the notion of architectural refinement are addressed, too. Section 4 draws some conclusions.

2 Static architectures of distributed systems

In this section, previous results on the modeling, specification and refinement of distributed systems will be summarized. The view taken here is that a system consists of the system architecture and the system behaviour. Elementary concepts are used to define these system aspects. For further details, see [Got93].

2.1 Elementary concepts

We start with the informal introduction of a very small number of elementary concepts. They form the starting point for the definition of several derived concepts. Due to their elementary nature, these concepts cannot be

formally defined. All that can be stated at this point is that these concepts are disjoint.

Definition (elementary concepts):

• An *agent* ag ∈ AG is a component performing actions.

• An *interaction point* ip ∈ IP is a conceptual location where actions may occur.

• An *action* (sometimes called *action occurance*) a ∈ Act is something that happens.

• Agent, interaction point, and action are disjoint concepts, i.e.,

$AG \cap IP = AG \cap Act = IP \cap Act = \{\}$

An action is performed by an agent or a set of agents, it may be internal or may occur at some interaction point or a set of interaction points. An agent thus is the carrier of actions, it can be characterized by its behaviour. This behaviour (a notion still to be defined) consists of actions local to the agent. Actions may also be non-local, such as interactions or transactions. Non-local actions may be performed by a set of agents and may occur at a set of interaction points. Interactions and transactions may also be considered as high-level actions, i.e., actions that can be decomposed into smaller units. Depending on what kind of action is taken as atomic on a given level of abstraction, the behaviour of a system can be characterized in different ways.

2.2 Derived concepts

With the elementary concepts agent and interaction point, more complex structures termed system architectures can be composed:

Definition (system architecture):

• A system architecture Arch is a structure <AG,IP,ArchF > where

- AG is a non-empty set of agents,
- IP is a set of interaction points, and
- ArchF: $AG \rightarrow 2^{IP}$ is a total function called architecture function associating with each agent a set of interaction points.

A set $AG^* \subseteq AG$ of agents has one or more interaction points in common if and only if $\bigcup_{ag \in AG^*} ArchF(ag) \neq \{\}$. We require as a rule of composition that a common interaction point is introduced explicitly into the architecture whenever a group of agents has the capability to interact directly. Depending on the kind of interaction, two or more agents may in general be involved in interactions. Whether such interactions will actually take place also depends on the behaviour.

Figure 2.1 shows a graphical representation of an architecture Arch1 formally specified in Table 2.1. From this architecture we can infer that ag_1, ag_2, and ag_3 have the capability to interact, however, we can not yet say whether they will actually do so.

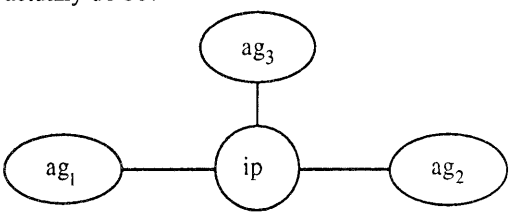

Figure 2.1: Graphical representation of architecture Arch1

This can only be derived from the behaviour of the agents and the interaction point.

$Arch1 = \langle \{ag_1, ag_2, ag_3\}, \{ip\}, ArchF \rangle$

$ArchF(ag_1) = \{ip\}$

$ArchF(ag_2) = \{ip\}$

$ArchF(ag_3) = \{ip\}$

Table 2.1: Formal specification of architecture Arch1

With the elementary concept action, more complex structures termed behaviours can be composed. A behaviour is specified as a conjunction of logical formulae, each expressing a restriction on the individual behaviour of a system component. The behaviour refers to externally visible actions, which are associated

with system components and therefore contain some reference to the system architecture.

Definition (system behaviour):

• Let Arch = ⟨**Arch**, **IP**, **ArchF**⟩ be a system architecture, Behavag be the behavioural specification of agents **ag** ∈ **AG**, and **Behav$_{ip}$** be the behaviour of interaction points **ip** ∈ **IP**. The system behaviour Behav is given as the conjunction of the component behaviour, i.e.,:

$$\mathbf{Behav} =_{DF} \bigwedge_{ag \in AG} \mathbf{Behav}_{ag} \wedge \bigwedge_{ip \in IP} \mathbf{Behav}_{ip}$$

Note that each system component is characterized by "local" properties. Non-local properties may be derived by logical reasoning. Having introduced system architectures and system behaviours, we can combine them into the derived concept of system.

Definition (system):

• A system $\mathcal{S}$ is a structure ⟨**Arch**, **Behav**⟩, where **Arch** is a system architecture and **Behav** is a system behaviour.

2.3 System Refinement

A notion of system refinement should take both system architecture and system behaviour into account. In general, it is desirable that the refinement of a single component (architecture and/or behaviour) has no influence on the other components. Only then will it be possible to perform incremental system design and modular verification, which is a prerequisite for the development of large systems. By incremental system design, we mean that we can modify or replace a part of the system without affecting the other parts. Modular verification means that only the modified or replaced parts have to be verified, not the entire system. To allow for incremental system design and modular verification, we have to make suitable restrictions with respect to architecture and behaviour.

A system $\mathcal{S}'$ refines a system $\mathcal{S}$, if $\mathcal{S}'$ is equivalent to or more specific than $\mathcal{S}$. With respect to the system architecture, this is the case if all agents and interaction points of $\mathcal{S}$ are represented in $\mathcal{S}'$, and if their composition is maintained in $\mathcal{S}'$. This is captured by the notion of architectural refinement below. Concerning the behaviour, $\mathcal{S}'$ is equivalent to or more specific than $\mathcal{S}'$ if the behaviour of $\mathcal{S}'$ logically implies the behaviour of $\mathcal{S}'$ mapped to the abstraction level of $\mathcal{S}'$. This mapping is expressed by a *representation function* **rep**.

Definition (system refinement):

• Let $\mathcal{S}$ = ⟨Arch,Behav ⟩ and $\mathcal{S}'$ = ⟨ Arch',Behav' ⟩ be requirement specifications. $\mathcal{S}'$ is a *refinement of* $\mathcal{S}$ *under the representation function* **rep** (written "$\mathcal{S}'$ refinesrep $\mathcal{S}$") if and only if the following is satisfied :

- **Arch'** *refines*$_{\text{Arch}}$ **Arch**
- $\models_i$ **Behav'** ⊃ **rep(Behav)**

With respect to architectures, we require that agents and interaction points be refined separately. In other words, a single component of the refinement (an agent or interaction point) is uniquely related to a single component of the refined architecture. Also, we require that the number of interaction points an agent is associated with remains the same. These and further architectural constraints can be formalized as follows:

Definition (architectural refinement):

• Let Arch = ⟨ AG,IP,ArchF ⟩ and Arch' = ⟨ AG',IP',ArchF' ⟩ be architectures. Arch' is an *architectural refinement* of Arch (written "Arch' *refines*$_{\text{Arch}}$ Arch") if and only if there is a refinement function $ref_{\text{Arch}} : AG \cup IP \rightarrow 2^{AG' \cup IP'}$ such that the following restrictions

hold:

- Each component of Arch is refined, i.e., ref_{Arch} is a total function.
- The refinement of an agent must include at least one agent. Formally:

$$\forall ag \in AG.\ ref_{Arch}(ag) \cap AG' \neq \{\}.$$

- The refinement of an interaction point must include at least one interaction point:

$$\forall ip \in IP.\ ref_{Arch}(ip) \cap IP' \neq \{\}.$$

- Each agent and each interaction point is refined separately, i.e., the refinement is disjoint:

$$\forall x,y \in AG \cup IP.\ (x \neq y \textit{ implies } ref_{Arch}(x) \cap ref_{Arch}(y) = \{\}).$$

- AG' is the set of exactly those agents resulting from the refinement, i.e.,

$$AG' = \left(\bigcup_{age \in AG} ref_{Arch}(ag) \cup \bigcup_{ipe \in IP} ref_{Arch}(ip) \right) \backslash IP'.$$

- IP' is the set of exactly those interaction points resulting from the refinement, i.e.,

$$IP' = \left(\bigcup_{ipe \in IP} ref_{Arch}(ip) \cup \bigcup_{age \in AG} ref_{Arch}(ag) \right) \backslash AG'.$$

- If an agent $ag \in AG$ is associated with an interaction point $ip \in IP$, then exactly one agent of the refinement of ag must be associated with exactly one interaction point of the refinement of ip. Formally:

$$\forall ip \in IP.\ \forall ag \in AG.\ ip \in ArchF(ag) \text{ implies;}$$

$$\left| ref_{Arch}(ip) \setminus AG' \cap_{ag' \in ref_{Arch} \bigcup_{(ag) \setminus IP'}} ArchF'(ag') \right|$$

and

$$\left| \{ag' \in ref_{Arch}(ag) \setminus IP' \mid ArchF'(ag') \cap ref_{Arch}(ip) \setminus AG' \neq \{\}\} \right| = 1$$

Figure 2.2 shows the graphical representation of a possible refinement Arch1' of the interaction point ip (compare Figure 2.1), which on a lower level of abstraction comprises an agent ag that can interact with ag_1, ag_2, and ag_3 through ip_1, ip_2, and ip_3, respectively. It is necessary to introduce interaction points in the refinement, because otherwise the rule of composition of architectures about their explicit introduction would be violated. Also, we notice that the duality between agents and interaction points is nicely carried into the refinement.

When we use temporal logic to characterize system behaviour, we require that a property must hold in the *initial* state of execution (properties holding throughout the execution can be defined using appropriate temporal operators). To express this formally, we use the notion of initial validity.

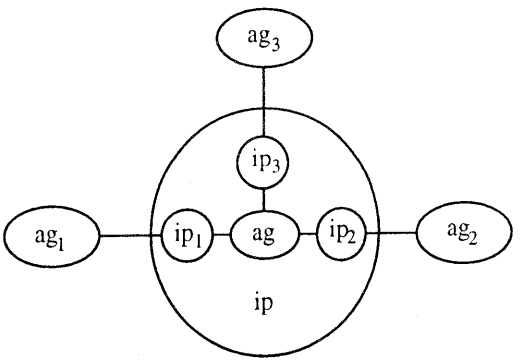

Figure 2.2: Graphical representation of the architectural refinement of Arch1'.

$Arch_1' = \langle$
$\{ag_1, ag_2, ag_3, ag\}, \{ip_1, ip_2, ip_3\}, ArchF' \rangle$
$ArchF'(ag) = \{ip_1, ip_2, ip_3\}$
$ArchF'(ag_1) = \{ip_1\}$
$ArchF'(ag_2) = \{ip_2\}$
$ArchF'(ag_3) = \{ip_3\}$

Table 2.2: Formal specification of architecture $Arch_1$

A formula ϕ is *initially-valid*, written $\models_i \phi$, iff ϕ is initially-valid in all models ([Got93]). Therefore, it is sufficient to require Behav' $\supset$ rep(Behav) to be initially-valid.

3 Dynamic architectures of distributed systems

In general, every part of the system architecture may undergo some dynamic evolution. E.g., the architecture function ArchF may be dynamically modified, thus changing the set of interaction points to which some agent is attached. Also, the sets AG and IP may be changed by creating or removing agents and interaction points, respectively. In the following, we will concentrate on the dynamical changes of the architecture function ArchF and the implications on the notion of architectural refinement. This seems to be sufficient to capture the dynamic evolutions of the Basic Reference Model for OSI and the ODP trading mechanism mentioned in Section 1.

3.1 Modeling and specifying dynamic architectures

Consider the Basic Reference Model (BRM) for Open Systems Interconnection ([ISO81]). In Section 1, we have quoted this document concerning the meaning of service-access-points. A service-access-point (SAP) can be interpreted as a specialization of the interaction point concept. Similarly, entities of the BRM specialize our notion of agent.

Thus, an architectural scenario as described in [ISO81] can be represented as shown in Figure 3.1a. This scenario may evolve to scenario 3.1b and then to 3.1c or back to 3.1.a during system execution. However, the scenario shown in Figure 3.1d must be considered illegal, since the (N)-service-access-point (N)-SAP is attached to more than one (N+1)-entity at a time.

The scenarios 3.1a through 3.1c can be formally specified as shown in Table 3.1. In the restricted architecture of the example, it is possible to enumerate all legal scenarios.

However, this is not feasible in general, therefore, other specification styles are needed.An alternative would be to characterize the legal scenarios by a set of properties, as shown in the formal specification in Table 3.2. These properties leave room for the dynamic evolution of the system architecture within certain limits.

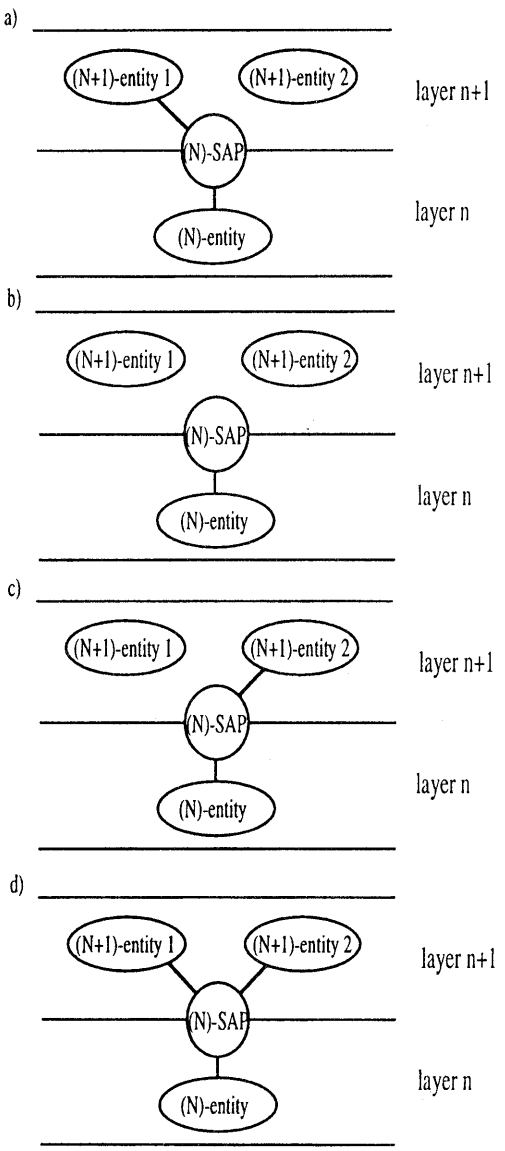

Figure 3.1: Graphical representations of architectural scenarios of the OSI BRM

E.g., at any moment in time, one of the agents (N+1)-entity 1 and (N+1)-entity 2 may be attached to (N)-SAP. However, it is excluded that both may be attached to (N)-SAP simultaneously.

$Arch_a$ = ⟨{(N+1)-entity 1,(N+1)-entity 2,(N)-entity},{(N)-SAP}, $ArchF_a$ ⟩
$ArchF_a$((N+1)-entity 1) = {(N)-SAP}
$ArchF_a$((N+1)-entity 2) = { }
$ArchF_a$((N)-entity) = {(N)-SAP}
$Arch_b$ = ⟨ {(N+1)-entity 1,(N+1)-entity 2,(N)-entity},{(N)-SAP},$ArchF_b$ ⟩
$ArchF_b$((N+1)-entity 1) = { }
$ArchF_b$((N+1)-entity 2) = { }
$ArchF_b$((N)-entity) = {(N)-SAP}
$Arch_c$ = ⟨ {(N+1)-entity 1,(N+1)-entity 2,(N)-entity},{(N)-SAP},$ArchF_c$ ⟩
$ArchF_c$((N+1)-entity 1) = { }
$ArchF_c$((N+1)-entity 2) = {(N)-SAP}
$ArchF_c$((N)-entity) = {(N)-SAP}

Table 3.1: Formal specification of architectures $Arch_a$ through $Arch_c$

□ ($Arch_{abc}$ = ⟨ {(N+1)-entity 1,(N+1)-entity 2,(N)-entity},{(N)-SAP},$ArchF_{abc}$ ⟩)
□ (∀ag ∈ {(N+1)-entity 1,(N+1)-entity 2}. $ArchF_{abc}$(ag) ⊆ {(N)-SAP})
□ (| {ag | ag ∈ {(N+1)-entity 1,(N+1)-entity 2} ∧ (N)-SAP Œ $ArchF_{abc}$(ag)} | ≤ 1)
□ ($ArchF_{abc}$((N)-entity) = {(N)-SAP})

Table 3.2: Property-oriented specification of architectures Archa through Archc

To emphasize that the architectural properties are invariant over time, we have applied the temporal operator 'henceforth'. Note that as a special case, static system architectures can be specified. E.g., (N)-entity will statically be attached to the interaction point (N)-SAP. Also, the sets AG of agents and IP of interaction points are required to remain unchanged during system execution according to the specification in Table 3.2.

Formally, we can capture the generalized notion of system architecture as follows:

Definition (dynamic system architecture):

• A *dynamic system architecture* $Arch_{dyn}$ is given as the conjunction of properties Archi, $1 \leq i \leq n$, such that for each moment in time, $Arch_{dyn}$ characterizes a system architecture as defined in Section 2.2:

$$Arch_{dyn} =_{Df} \bigwedge_{1 \leq i \leq n} Arch_i$$

It should be pointed out that this definition of a dynamic architecture not capture the evolution of the architecture completely, but gives a number of restrictions that the architecture must fulfil at any moment in time. It is, e.g., still possible that the system architecture is a static one. The actual evolution is a behavioural aspect and therefore has to be specified as part of the system behaviour. Also, it is still left open whether an agent is allowed to attach or detach itself to or from some interaction point, or whether some distinguished agent is responsible for all architectural modifications.

3.2 Refinement of dynamic architectures

As a consequence of the generalization of system architectures, it is necessary to reconsider and generalize the notion of architectural refinement for static architectures introduced in Section 2.3. According to that notion, Arch' $refines_{Arch}$ Arch, if all agents and interaction points of Arch are represented in

Arch', and if their composition is maintained in Arch'. How can this be extended to dynamic architectures?

A first observation we can make here is that the restrictions on the dynamic architecture Archdyn have to hold in the refined architecture Arch'dyn, too. In other words, the architecture $Arch'_{dyn}$ has to be equivalent to or more specific than $Arch_{dyn}$ with respect to some refinement mapping. Formally, we can express this requirement as follows: $\models_i Arch'_{dyn} \supset rep(Arch_{dyn})$

i.e., the properties characterizing $Arch'_{dyn}$ logically imply the properties characterizing Archdyn when mapped to the abstraction level of $Arch'_{dyn}$. As it turns out, this is indeed a necessary, but not a sufficient condition. To see why this condition is not sufficient, consider the specification Archabc in Table 3.2. With the identity mapping ref_{Arch}, $\models_i Arch_{abc} \supset rep(Arch_{abc})$ holds. However, at a given moment in time, we could have the abstract architecture shown in Figure 3.1.a, while the refinement is represented by Figure 3.1c. Therefore, a stronger condition ruling out this possibility is required.

The crucial observation here is that the dynamic architecture Arch'dyn has to refine $Arch_{dyn}$ *at all moments in time*. Formally, this can be expressed as follows:

$\models_i Arch'_{dyn}$ $refines_{Arch}$ $Arch_{dyn}$

This condition is somewhat stronger than the first one, because it additionally requires that if an agent ag of $Arch_{dyn}$ is associated with an interaction point ip at any given moment in time, then exactly one agent of the refinement of ag must be associated with exactly one interaction point of the refinement of ip at that moment in time.

A drawback here is that the second condition is more difficult to prove. The difficulty arises from the additional condition about associated interaction points. This is not expressed by the architectural properties alone, but is a behavioural aspect that can only be derived from the system behaviour.

This leads us to the following notion of system refinement:

Definition (system refinement):

• Let $\mathcal{S} = \langle Arch, Behav \rangle$ and $\mathcal{S}' = \langle Arch', Behav' \rangle$ be requirement specifications. $\mathcal{S}'$ is a refinement of $\mathcal{S}$ under the representation function rep (written "$\mathcal{S}'$ refinesrep $\mathcal{S}$") if and only if the following is satisfied:

- $\models_i$ Arch' $refines_{Arch}$ Arch
- $\models_i$ Behav' $\supset$ rep(Behav)

4 Conclusions

In this paper, we have presented some preliminary results on modeling, specifying and refining dynamic system architectures. We have considered the evolution of interaction point associations, as e.g. addressed by the OSI Basic Reference Model or the ODP trading concept. It turned out that our definition of a dynamic architecture does not capture the evolution of system architecture completely, but has to be augmented by an appropriate system behaviour. This aspect needs further study. Also, we have extended the notion of architectural refinement to cover dynamic changes. Some criteria have been identified and formalized that capture our intuition about such refinements. Case studies will have to show the practicability of these criteria when proving the correctness of distributed systems with dynamic architectures.

Ackowledgment:

This work is sponsored by the International Scientific Joint Research program of the Ministry of Education & Science and Culture of Japan.

References

[Got93] Gotzhein, R.: Open Distributed Systems - On Concepts, Methods and Design from a Logical Point of View, Vieweg Wiesbaden, 1993

[ISO81] ISO/TC97/SC16: Data Processing - Open Systems Interconnection - Basic Reference Model, Computer Networks 5 (1981), pp. 81-118

[ISO93] ISO/IEC JTC1/SC21/WG7 N6084: Working Document on the Trader (ODP Trader), July 1993