Tempo:
Defining Software Processes in an Approach Based on Objects with Roles

Noureddine Belkhatir
LGI
BP 53
38041 Grenoble France
e-mail: belkhatir@imag.fr

Walcélio L. Melo
University of Maryland
UMIACS,
College Park, MD, 20742 USA
e-mail: melo@umiacs.umd.edu

Published in the *Proc. of the 1st Int'l Conf. on Object-Role Modelling*.
Magnetic Island, Australia, July 1994.

Abstract

Recent developments have shown the need for an integrated view of the large-scale software development environment that takes account of how software products are managed and produced. To fulfill these requirements, the activities carried out during software development must be managed. In this paper, we describe the work we are carrying to support software process definition and enacting.

We shall discuss an approach to process modeling known as an object-oriented software process modeling language. Special attention is paid to how object-oriented concepts, the role *concept, and trigger rules can be integrated to describe software process models.*

1 Introduction

There is broad agreement that software engineering environments (SEE's) should provide explicit support for capturing and controlling software processes. Software processes should be represented explicitly and, in part, automatically executed by a SEE [16]. To satisfy this requirement, several research programs have explored data integration and centralized control using integrating platforms. These platforms provide support for product structuring, versioning, software configuration, and other engineering processes. On the basis of this experience, the entity-relationship-attribute data model (ERA), extended with Object-Oriented (OO) concepts, has been successful used as a conceptual framework for process definition. Nevertheless, although the basic OO concepts are a good starting point, they alone are not sufficient for capturing all the complexity of software engineering processes and their evolution. We claim that major capabilities such as *multiple object behavior* and *modeling of activities with long duration*, including long events and the time concept, need to be added to such a conceptual framework.

We have addressed these problems in our work in the framework of the Adele project. Adele, which was initially a revision control system, has been extended with user-defined entities and relation types to support the definition and control of large software systems. To make it possible to support software processes, the Adele language has been extended with event-condition (ECA) rules, and the Adele kernel has been extended with a trigger mechanism. This new system is called Adele 2 [2]. Adele 2 is now a commercial software product and is used for automatic software configuration activites in various European companies, such as Matra-Space, and ESPRIT projects, such as REBOOT. On the basis of our experience working with Adele 2, we have concluded the O.O. data model, even when integrated with ECA rules, is not sufficient for defining software process models. Once ECA rules are fragmented among data and relation types, it becomes difficult to control process enacting and to manage changes in processes. To overcome such drawbacks, we have initiated the Tempo project [4] to support software development processes. Tempo has enable an accurate set of software activities to be aggregated in process types and the static and behavioral description of objects, manipulated by such activities, to be re-defined according to their **roles** in a process step. As time plays an important role in any SEE, because we deal with long transactions, we have also studied how temporal information can be represented and how it can be exploited in the software process evolution. As we shall show later, we have decided to extend our ECA formalism with temporal logical operators. We have modified the Tempo process engine to make it possible to interpret temporal ECA rules. This paper discribes these extensions and shows examples of how they are used for modeling, evolving, and enacting software processes.

1.1 Rationale

Adele/Tempo is a Process-Oriented Software Development Environment (POSE) [5] that focuses on the following capabilities:

- management of resources shared by a team for enforcing cooperative work, by providing objects with roles;
- activity coordination and traceability of activity execution, by providing activity management.

1.1.1 Object roles

Using object-oriented technology, we can model software process steps (or sub-processes) using complex active objects. As software objects associated with sub-processes also provide operations (methods), the combination of these two kinds of facilities could be used to describe statically (process model) and dynamically (process enactment) the software processes of a particular environment or company.

As the only structuring concept supplied by the traditional O.O. paradigm is the classification concept, however this model supports only one object behavior description, which can be refined by specialization using the class structure. All applications are supposed to conform to in that structure and consequently to that behavior. This mono-behavioral belief, which reigns in the O.O. world, impacts directly on the difficulties involved in process management. If we allow an object to behave differently depending on where, when, and how is it used, and enable it to be seen through different prisms, we claim it will be possible to better manage process changes. Using this approach, we could modify software processes by creating new ways to see and manipulate already-instantiated objects in harmony with old definitions.

1.1.2 Activity management

In an O.O. approach, objects interact by executing and exchanging messages via methods that support the active part. A number of mechanisms have been suggested for controlling and synchronizing interaction between methods. The most influential is the trigger mechanism based on Event-Condition-Action rules. ECA rules have been proposed to manage communication by extracting actions specific to method driving from method definition.

The trigger mechanism is most integrated with mechanisms supporting only short transaction (in the sense of dabase management systems). In the framework of software process, however activities have a long duration (long transactions). Thus, to coordinate method execution and control activity evolution, we need mechanisms that keep track of activity chaining and trace its execution. Therefore, we must introduce concepts and mechanisms for dealing with *time*. These requirements lead to:

1. a log database that contains capabilities for tracing object states;
2. capabilities to reason with temporal events.

1.2 Outline

Section 2 presents an overview of the Adele/Tempo system. Section 3 presents an overview of the Tempo software process modeling language. We present our conclusions in section 4.

2 An overview of Adele/Tempo

The Adele/Tempo system consists of three basic parts (see figure 1):

- A resource manager using Adele database as a persistent object base for storing objects and activities and for tracing the project's progress. ADL-DB supports an entity-relationship data model which is extended with object-oriented concepts like inheritance, methods and encapsulation. Simple and composite objects with attributes and relationships can be described and managed. This component of Adele/Tempo architecture is responsible for the data integration according to conceptual model proposed by [20].
- An activity manager which is the responsible for the control integration in our platform. This activity manager is driven by Temporal-event-condition-action rules (TECA) and supported by a trigger mechanism.
- A process manager which offers definition concepts for activity structuring by the process and role concepts. Process occurrences are supported by work environments (WE) wherein software activities are performed. The process manager, based on the activity manager, manages communication and synchronization between teams and between agents involved in a same project. It also controls the consistency of complex objects used simultaneously in different work environments by different software processes occurrences and agents. This component represents the conceptual component responsible for process integration in the Adele/Tempo architecture.

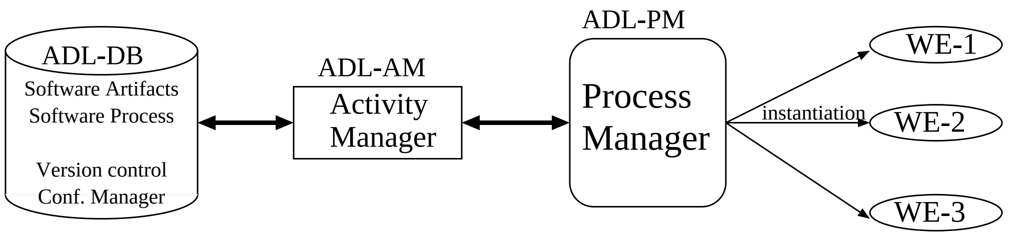

Figure 1: The kernel of Adele/Tempo environment.

3 The Tempo software process modeling language

Tempo [4] is an executable formalism for describing and enacting software process models. It uses an object-oriented approach extended by the addition of a multi-behavioral facility. The multi-behavioral facility is a major problem currently being researched in a wide range of fields. The problem arises when developing large, complex systems characterized by the presence of several agents, working on shared resources and using multiple representations and multiple development strategies. In this context we need a way of expressing relationships between multiple view points.

There are three sides to our approach:

- modeling of software activities by software process types;
- analysis of the various view points and software component life cycle states using the role concept;
- describing software temporal constraints by temporal-event-condition action rules (triggers rules). ECA rules are extended, using a temporal modality, to support long transactions (long duration activities). The temporal modality is applied to events, and it allows reasoning in relation to past activities.

3.1 The Adele data model

The Adele data model is derived from an entity-association model and integrates object-oriented concepts [2]. The basic entities of the model are object type and relationship type. Each entity (object and relationship) possesses static (attributes) and dynamic (methods, event-condition-action temporal rules) properties. Relationships are binary.

The data model supports complex objects referred to as aggregates. An aggregate is an object linked to its components by relationships. For example, a Pascal module can consist of an interface and an implementation. Consequently, the Pascal module object can be represented as an object linked to two other objects by two types of relationship, possesses-interface and possesses-implementation. Aggregate semantics are defined by the dynamic properties of the relationship linking the aggregate to its components. The semantics are defined by the user; any aggregate can thus be defined using its own semantics and consistency constraints.

In Adele, a type is defined by an interface part, and an implementation part which describe type instance properties.

The notions of interface and implementation are similar to those used in programs written using languages such as ADA and MODULA, or indeed certain object-oriented languages.

The interface part contains the type properties that are visible and are exported. The implementation part contains private properties and the implementation of visible methods.

3.2 Software process types: modelling software process models

TEMPO describes and executes software processes. A software process model of considerable size may thus be written by a group of various software process types. A software process type can aggregate other software process types.

For example, an activity to check a module design document comprises two sub-processes:

1. A sub-process that models the modification activity that makes changes to the design document.
2. A sub-process that models the revision activity that approves design document modifications that have been made.

```
MonitorDesign ISA PROCESS;
   CONTROL md;
      sub = ModifyDesign;
   CONTROL rd;
      sub = ReviewDesign;
END_OF MonitorDesign;
ModifyDesign ISA PROCESS;
   ATTRIBUTES
      begin_date = DATE := now();
      end_date = DATE;
                deadline = DATE;
   METHODS . . .
   RULES  . . .
END_OF ModifyDesign;

ReviewDesign ISA PROCESS; ...
```

The example above shows the software process type `MonitorDesign`, composed of the sub-processes `ModifyDesign` and `ReviewDesign`. The activity coordinating the module design document modification is represented by the `MonitorDesign` type. `ModifyDesign` is the type which describes the design document modification process, and `ReviewDesign` is for revising this modification.

It is possible, for every process type, to define attributes, methods, and temporal constraints by using the event-condition-action rules.

3.3 The temporal contraints

Due to the long life duration of software processes, we also need management mechanisms for time constraint and traceability. To support process evolution, we also need to be able to reason about execution sequences. Thus we have introduced temporal reasoning capability in Adele/Tempo system to plan and schedule activities in software process. Temporal constraints can be used either to schedule activities or to aid synchronization and cooperation between activites. To integrate temporal constraints in the Tempo language, we are incorporating temporal features in the event-condition-action rules (ECA). After we have extended the trigger mechanism of the Adele/Tempo system to support reasoning about time. This new formalism combines of standard ECA rules and temporal logic predicates. In this way, we are adding a new dimension to the Adele/Tempo system for managing process evolution using temporal knowledge in the field of software process management.

3.3.1 Temporal event-condition-action rules

Temporal contraints are described by temporal-event-condition-action (TECA) rules. TECA rules are simIlair to Alf/Pcte [6], Damokles [7], and HiPAC [11] trigger rules. Interpretation and execution of these rules are based on trigger mechanism integrated to the object management system of Adele/Tempo [2]. For example:

```
ModifyDesign ISA PROCESS;
   ATTRIBUTES
      begin_date = DATE := now();
      end_date = DATE;
      deadline = DATE;
   METHODS
      continue_execution;
      . . .
   RULES
(1)   AFTER WHEN deadline_arrived
            DO stop_execution;
(2)   PRE WHEN continue_execution
          IFPAST not deadline_changed
          FROM last(deadline_arrived) UNTIL now()
          DO ABORT;
END_OF ModifyDesign;
```

1. The rule described in line 1 specifies the design document modification activity must stop when the date foreseen has been reached.

2. The rule in line 2 states that resumption of the activity (it hasn't been completed yet) first requires the termination date be changed.

3.3.2 TECA rules execution module

TECA rules are defined in the data model (not shown in this article) and in the software process module. They are inherent in the hierarchy of object types and software processes. In the data model, the TECA rules describe integrity limitations that are independent of the object's usage context. On the other hand, these rules are used to express the software development strategy used in the software process model: order of activity execution, activity synchronization, and software resource usage limitations.

A TECA rule is expressed in the following manner: "WHEN temporal-event DO Method", where "temporal-event" is the temporal predicate expressing:

1. an event in the present state of the developpement environment (e.g., objects, tools and agents states) or
2. an event about the past state of the developpement environment which has been stored in the database of the Adele/Tempo system.

Method is an instruction sequence.

```
DEFEVENT delete_obj = [ !cmd = rmobj] ;
```

The `delete_obj` event is defined in this example as being the event that survives whenever the current command (`!cmd`) is an object removal command (`rmobj`).

A method is a program written in simple, direct language similar to the Unix shell.

```
METHOD delete ;
       IF [state = stable] THEN ABORT
       ELSE "rmobj %name ";
END delete;
```

This method allows for object removal in an unstable state.

Triggers rules can be defined to control the execution of methods. Some triggers will be executed before the methods, acting as pre conditions, others after the method execution, as post-conditions. Since triggers are (originally) intended to enforce consistency, any inconsistency found by a trigger must be able to undo (roll-back) the method execution. Thus for any method the following instructions will be executed:

```
PRE list of triggers
     Action (Method)
POST list of triggers
```

The whole execution is always a single transaction, even if the triggers or the methods send messages to other objects. The execution of a primitive `ABORT`, anywhere in a block (`PRE/Action/POST`) will undo everything that was done in this block. `AFTER` triggers are executed after transaction committing; they are used to execute actions when sure that the transaction succeeded, as for example sending notifications, or to execute new actions whose failure must not undo the main action.

If the transaction failed, `ERROR` triggers are executed.

Thus for each object and relation type, there are five blocks:

```
PRE   list of triggers
```

```
          METHOD list of methods
     POST  list of triggers
     AFTER list of triggers
     ERROR list of triggers
```

3.3.3 Related work

Other SEEs use ECA rules, such as AP5 [12], Alf/Pcte [6] and Appl/A [19]. Alf and Tempo provide four TECA rules execution modes (PRE,POST,AFTER and EXCEPTION) whereas AP5, Marvel, Triad and Appl/A support only one mode of execution, the mode POST. All these systems do not provide concepts for handling Temporary constraints.

Alf's event-condition-action rules are similar to those offered by Adele/Tempo. However, Alf does not have the concept of method. Such as in Marvel, all the actions must be defined by the operators (production rules according to the MASP formalism). The execution of an operator can trigger a forward chaining process in the user's private space (ASP in the Alf terminology). ECA rules are defined elsewhere and executed by another mechanism called "trigger". In Tempo, we adopted only one concept to define both the constraints on the execution of methods and the constraints on the utilization of objects, i.e, the TECA rules.

Some other systems like AP5 [12] and ODE [9] also, in a way, provide ECA rules concerning time. However, these systems do not allow the specification of conditions about past actions. They limit themselves to specifying that for example, some actions (mainly methods) must be executed at an absolute/particular time in the future, for example, every day at 9 a.m or tomorrow at 6 p.m. etc.

3.4 Object with roles

Multiple perspectives or view points often occur during a software product life cycle. Several users treat objects concurrently, using different views of the objects with limited, controlled actions specific to their activity. These users, controlled by multiple development strategies, handle different models of the same product. A SEE should provide a work environment that can describe and control these various aspects.

Owing to the role concept, Tempo language allows each software process occurrence to have local contraints and properties for each object treated [3].

Roles have a defined type. A role type may reference different types of objects. This strategy allows the integration of various types of behaviour and properties, coming from different types of objects, within a single perspective. This strategy unifies thus the treatment of a heterogeneous set of objects. The advantage of this approach is that a set of object types with different static and dynamic characteristics can be viewed, using the role concept, during a specific software process execution step in a coherent, homogeneous fashion. This coherence is maintained by using the multiple heritage rules available in object-oriented models. The principal difference is based on the extent of the roles. At the definition level, a role type is viewed as the specialisation of the types it contains. However, at the instance level:

1. Objects created from a role are not included in the role's specialised type extensions.
2. A subset of objects pertaining to these types may belong to the role.

A software process type may have several role types; a software process becomes a list of roles whereby each object type may have different roles. Consequently, two objects of the same type may be controlled differently within the same software process. At the same time, an object can play roles within different software processes. For example:

```
ReviewDesign ISA PROCESS;
   ROLE under_review;
        derived_from = specification_document;
```

```
          . . .
    ROLE requested_change;
         derived_from = cc_request;
                 . . .
END_OF ReviewDesign;
```

3.5 Role discussion

One may claim that this kind of contextual behavior can be achieved by standard object-oriented techiniques. Roles and type look similar. This raises the question: Can roles be implemented in terms of typing and sub-typing? Is the concept of role needed? We claim the role concept has the following properties:

1. Prevent type explosion.
   A role, as well as a type, is a template applied to a set of instances sharing the same definition (static and behavioral). A given object instance can be simultaneously a member of different roles (classes). Both roles and types can be seen as a viewing mechanism since a given object instance has a different description depending on the role (class) from which it is managed. One would need to create a sub-type for all the possible combinations of roles for a single type, and to change instance type dynamically each time a new role is applied to it. However, there is a fundamental difference:

   > The association between an instance and its type is defined statically at instantiation time, while an instance can be bound dynamically to an arbitrary role at any time.

   Furthermore, since the instance can be share and play different roles simultaneously, dynamic typing cannot be used. We introduce the possibility of changing type dynamically. In an OO system the type definition is created first, and then the instances of the types. In the Adele/Tempo system, on the other hand, the instances usually are created first, and are associated dynamically, for a while, to a (set of) role(s).

2. Identity is not altered.
   Since a given object can be simultaneously a member of different roles, there are compatibility rules between the roles allowed for shared objects. In TEMPO, objects can change behavior depending on the context without changing identity.

3. Schema evolution-version.
   Schema evolution support is an important facility for a software engineering environment, because we need to make it possible to evolve the characteristics of software objects manipulated during the software processes. The role concept naturally integrates a type evolution facility, since role types are similar to object types in O.O. languages. The role concept offers two kinds of evolution:

   (a) role definition can change generating role version;

   (b) objects can change their roles dynamically.

3.6 Related work

Other SEEs have recently integrated concepts similar to the role concept proposed in Tempo, for example, ES-TAME [13] and Alf/Pcte with its "*Work Scheme*" concept [6].

ES-TAME allows dynamic change of object type during the execution of software activities. As a result, object attributes and methods can change depending on the activity under which this object is handled. This flexibility of type modification can be taken as the implementation mechanism for the role concept proposed in Tempo.

In Alf/Pcte [6], the same object can have different properties (attribute and relations) depending on the "Work Scheme" under which it is handled. Altough Alf/Pcte offers also trigger rules [14], but time dimension is not taken into account.

In the field of databases, the problem related to the modelling of object roles tend to be used as a means of improving the description of object evolution phases or object utilization facets. Several studies have been and are still on this problem. In general, the strategy used is to adopt a persistent object-oriented language and then extend it with the view concept, like the Aspect languages [17], Fibonacci [1] and Views [18]. Object handling is therefore carried out via its view. The limits presented by these approaches as compared to ours are the following:

- They offer no concepts for modelling activities where objects can be handled. Therefore, the way in which an object is perceived and handled is described without considering the context in which the said object will be used.
- The choice of the use of a view is left to the application's programmer. This decision is based on the description, in the program, of the view that must answer to the messages sent to the object.
- In general,these languages, do not offer concepts for aggregation of different points of view.

4 Conclusions

The Tempo language is designed for describing software processes and, in particular, for enforcing cooperative work.

The main capabilities of Adele/Tempo system are:

- The model used to describe processes is an object-oriented model. Each process step occurrence aggregates a set of entities. Each entity is managed by Adele database. When an entity is manipulated it is considered by Tempo as process resource. A resource "plays a role" in a software process step. The role concept makes it possible to customize the characteristics and behavior of a resource. Resource attributes can be forbidden, modified, created, and overloaded to satisfy the requirements of a process step. In the same way, resource behavior can be tuned, and specific communication and synchronization operations can be described take account of events generated inside/outside a software process step.
- Adele/Tempo provides new facilities for working with object log and process evolution. The trigger mechanism is under extension to support temporal event interpretation, i.e. event-condition-actions rules can be specified with temporal logic predicates. In this way, long transactions can be controlled more successfully. Rules can verify objects manipulation and evolution by analyzing operations performed on such objects during the software process life.

We believe the unification of these features contributes to improved cooperative work in a team of developers and proper sharing of resources among a set of processes. The management of software process evolution is accomplished by (a) making it possible for an object, already instantiated, to change its behavior and static properties dynamically using the **role** concept, (b) following object evolution in time by a log database, and (c) controlling such evolution by temporal ECA rules.

Acknowledgements

During this work, W. Melo was supported by the Technological and Scientific Development National Council of Brazil (CNPq) under grant No. 204404/89-4.

The authors would like to thank one of the anonymous referees for his valuable comments and suggestions.

References

[1] A. Albano, R. Bergamini, G. Ghelli, and R. Orsini. An object data model with roles. In R. Agrawal, S. Baker, and D. Bell, editors, *Proc. of the 19th Int'l Conf. on Very Large Data Bases*, pages 39–51, Dublin, Ireland, August 24–27 1993.

[2] N. Belkhatir, J. Estublier, and W. L. Melo. Adele 2: a support to large software development process. In Dowson [8], pages 159–170.

[3] N. Belkhatir, J. Estublier, and W. L. Melo. Software process model and work space control in the Adele/Tempo system. In Osterweil [15], pages 2–11.

[4] N. Belkhatir and W. L. Melo. Tempo: a software process model based on object context behavior. In *Proc. of the 5th Int'l Conf. on Software Engineering & its Applications*, pages 733–742, Toulouse, France, December 7–11 1992.

[5] N. Belkhatir and W. L. Melo. Supporting software maintenace processes in Tempo. In *Proc. of the Conf. on Software Maintenance*, pages 21–30, Montreal, Canada, September 1993. IEEE Press.

[6] J.-C. Derniame, C. Godart, V. Gruhn, and J. Lonchamp. Process-Centered IPSEs in ALF. In N. H. Madhavji G. Forte and H. A. Muller, editors, *Proc of the 5th Int'l Workshop on Computer-Aided Software Engineering (CASE'92)*, pages 179–190, Montréal, Québec, Canada, July 6–10 1992. IEEE Computer Society Press.

[7] K.R. Dittrich. The Damokles database system for design applications: its past, its present, and its future. In K. H. Bennett, editor, *Software Engineering Environments: Research and Practice*, pages 151–171. Ellis Horwood Books, Durhan, UK, 1989.

[8] M. Dowson, editor. *Proc. of the First Int'l Conf. on the Software Process*, Redondo Beach, CA, October 21–22 1991. IEEE Computer Society Press.

[9] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. In volume 21, no. 2 of *ACM SIGMOD Record*, pages 81–90. ACM Press, June 1992.

[10] F. Long, editor. *Proc. of Int'l Workshop on Software Engineering Environments*, volume 467 of *LNCS*, Chinon, France, September 18–20 1989. Springer-Verlag, Berlin, 1990.

[11] D. R. McCarthy and U. Dayal. The architecture of an active database management system. In *Proc. of ACM SIGMOD 89*, pages 215–224, Portland, OR, May 1989.

[12] K. Narayanaswamy. Enactment in a process-centered softwre engineering environment. In W. Schafer, editor, *Proc. of the 8th Int'l Software Process Workshop*, Germany, 1993. IEEE Computer Society Press.

[13] M. Oivo and V. R. Basili. Representing software engineering models: the TAME goal oriented approach. *IEEE Transactions on Software Engineering*, 18(10):886–898, 1992.

[14] F. Oquendo, G. Boudier, F. Gallo, R. Minot, and I. Thomas. The PCTE+'OMS: A software engineering database system for supporting large-scale software developpement environments. In *Proc. of the 2nd Int'l Symp. on Database Systems for Advanced Applications*, Tokyo, Japan, April 1991.

[15] L. Osterweil, editor. *Proc. of the 2nd Int'l Conf. on the Software Process*, Berlin, Germany, February 1993. IEEE Press.

[16] L. J. Osterweil. Software processes are software too. In *Proc. of the 9th Int'l Conf. on Software Engineering*, pages 2–13, Monterey, CA, March 30-April 2 1987.

[17] J. Richardson and P. Schwartz. Aspects: Extending objects to support multiple, independent roles. In *Proc. of the Int'l Conf. on Management of Data*, volume 20 of *ACM SIGMOD Record*, pages 298–307, May 1991.

[18] J.J. Shilling and P.F. Sweeney. Three steps to view: Extending the object-oriented paradigms. In *Proc. of the OOPSLA'89*, volume 24, no. 10 of *ACM SIGPLAN Noticies*, pages 353–361, Oct. 1989.

[19] S. M. Sutton, D. Heimbigner, and L. J. Osterweil. Language constructs for managing change in process-centered environments. In , volume 15 of *ACM SIGSOFT Soft. Eng. Notes*, pages 206–217, Irvine, CA, 1990.

[20] A. I. Wasserman. Tool integration in software engineering environments. In Long [10].