# Graphics Tools in Adagio, A Robotics Multitasking Multiprocessor Workstation

Stephen A. MacKay
Peter P. Tanner†

Laboratory for Intelligent Systems
National Research Council of Canada
Ottawa, Ontario K1A 0R8 Canada.

## Abstract

The development of Adagio, a robotics simulation workstation, has involved the implementation of several techniques unique to the system. Based on the message-passing, multitasking multiprocessor realtime operating system Harmony, Adagio is programmed using a large number of cooperating tasks. Several techniques are based on the concept of a server, a task that is alone responsible for governing a scarce resource. The Graphics Server task, the Data Structure Server task, and the Tracker Server task are responsible for the management of the frame buffer, the 3D geometric data structure and the screen tracker, respectively. Each of these servers is then a separate tool necessary for the implementation of the whole system. Each runs in parallel with other tasks and can handle requests for service from any task.

## Résumé

Le développement d'Adagio, station de travail dédié à la simulation de robots, a conduit à l'implémentation de techniques originale. Basé sur le système d'opération de temps réel, multi-tâche, multi-processeur Harmony (lui-même basé sur le transfert de messages), Adagio a été programmé en utilisant un grand nombre de tâches coopérants entre-elles. Plusieurs de ces techniques sont basées sur le concept de serveur (le serveur est la seule tâche responsable d'une certaine ressource). Les tâches "Graphics Server", "Data Structure Server", et "Tracker Server" gèrent respectivement la mémoire d'image, la base de données géométriques 3D, et le curseur d'écran. Chacun de ces serveurs est donc un outil séparé, nécessaire à l'implémentation du système complet. Chacun s'exécute en parallèle avec d'autres tâches et peut gérer des demandes venant d'autres tâches.

**Keywords:** robot simulation, realtime, multitasking, multiprocessing, message passing, server task, frame buffer, windows, 3D geometric data, user interfaces, screen tracker.

† current address: Computer Graphics Laboratory
University of Waterloo
Waterloo, Ontario N2L 3G1 Canada.

NRC number : 25461

## Introduction

Adagio is a robotics simulation workstation currently under development at the National Research Council. The workstation, when completed, will give the user the capability of creating and manipulating 3D objects in a robot environment, of specifying the robot task, and of viewing the results of a robot simulation. As such it will certainly be usable for many other applications that can make use of a 3D window-based near realtime display with extensive interaction capabilities.

The use of the Harmony operating system, a multitasking multiprocessor realtime message-passing system, as a base, has led to different approaches to the software architecture of an interactive graphics system. This paper discusses several of these approaches.

Three servers, a Graphics Server, a Data Structure Server, and a Tracker Server follow an idea common in multitasking systems; i.e. each is solely responsible for a specific scarce resource. The Graphics Server is charged with the maintenance of the frame buffer, while the Data Structure Server maintains the 3D geometric representation of the robot and its environment. The Tracker Server communicates with the Tablet Server to provide continuous tablet tracker echoes. It is particularly well suited for a multiwindow system and provides richer user feedback than is currently available on most systems.

## Adagio Overview

### Goals

Adagio [Tann85b] is a workstation being developed to support research in intelligent robotics. It is intended to provide a simulation facility for studies in the off-line programming of sensor-based robots. The functional requirements of providing the user with a view of the current status of the robot in its environment with near realtime updating (i.e. 5-30 frames per second), and providing for rich interactive dialogues for experiments in interactive graphics-based robot programming, led to the use of a powerful frame buffer display (in our case, an Adage 3000 graphics system) with a window-based user interface. Special software for the Adage 3000 bitslice microprocessor has been written to support multiwindow near realtime line-drawing and polygon faceted 3D renderings of a single scene.

Another goal has been to take advantage of the multitasking inherent in the Harmony operating system to improve various aspects of interactive graphics systems. This has resulted in implementing a highly parallel base for user interaction [Tann85a], a switchboard input model [Tann86], and the various tools described in this paper.

### Harmony

The Adagio system design is influenced by the architecture of Harmony, a multitasking realtime operating system with rapid inter-task message passing developed at the National Research Council of Canada [Gent85]. A few details of its properties are given here in order to aid in the appreciation of the multitask design presented in the paper.

Programs written for a Harmony-based system tend to be implemented as a set of many small tasks. A task is often used as one would use a subroutine in a conventional operating system, except that an instance of the task must be explicitly created, and once it has been created it executes independently of, and in parallel with, the task that created it. Task primitives are relatively cheap, message communication takes little time, (a send-receive-reply sequence takes about one millisecond), and the creation and destruction of tasks are inexpensive. Tasks can be created and destroyed as needed, and often are quite small with short lifetimes.

The communication and synchronization of tasks, used extensively throughout this workstation design, is based on the send-receive-reply mechanism provided by Harmony. A task may send information or a request for information to another task by issuing the _Send command, passing a variable-length message. If the recipient task is alive, the sending task then blocks until a reply arrives from the recipient or the recipient is destroyed. A task receives a message by issuing a _Receive command, which blocks until a message is received, or a non-blocking _Try_receive command. In either case, the ID of the sending task is returned to the recipient task (0 if no message was waiting in the case of _Try_receive) along with a copy of the message that was sent. The _Receive or _Try_receive command may specify that messages are to be received from a specific task, or from any task. A task that has received a message from another task may reply to the sending task with the _Reply command. The _Reply command unblocks the sending task and causes a variable-length message to be replied to it. A task need not reply to a sending task immediately, replies may be issued at any time and in any order necessary to achieve synchronization.

The send-receive-reply paradigm also encourages the use of *server* tasks, based on the *administrator* concept [Gent81], to perform various duties for other *client* tasks, such as the managing of scarce resources. A typical server never sends messages, it only receives and replies to requests. It often will have one or more *worker* tasks doing the time consuming work so that the server can respond to requests as quickly as possible. Because most servers do not send messages, two servers that must communicate usually do so through a *courier* task created for the purpose. A courier alternates between sending a request for information to one task and sending the resulting information to the other. The Graphics Server, the Data Structure Server and the Tracker Server described below are three examples of servers used in Adagio.

### Hardware Configuration

The Harmony operating system is particularly suited for running on a multiprocessor. Such a machine, a Chorus multiprocessor, consists of three single board computers, using Motorola 68000 processors on a Multibus backplane. Providing the graphics processing for the workstation is an Adage/Ikonas 3000 graphics system, a powerful frame buffer display with a 32-bit bitslice processor, a single 68000 also running Harmony, and an image memory 1024 by 1024 by 24 bits (512 by 512 visible).

### Interaction Model

Adagio's design is based on the concept of a *switchboard* [Tann86]. The Switchboard, shown in Figure 1, is a server that corresponds with a number of input device servers, through couriers, and a number of client tasks that make use of the values from these devices. The client tasks request input from the Switchboard, which in turn routes to these tasks input from devices to which they are connected. A flow of messages is thus established going back and forth between the producers and the consumers of input.

### Graphics Server

The Graphics Server, shown in Figure 2, replaces the graphics subroutine support package traditionally used in a single-task graphics program. Running as an independent task with the role of managing the frame buffer and the Adage bitslice processor, this server handles three types of request messages - window manipulation messages, 2D graphics messages, and 3D graphics messages.

### Screen Windows

Screen windows in Adagio are implemented in a manner different from those of many other window based systems [Tann86]. All windows are tightly coupled, assisting in the single job of creating and manipulating a data structure defining the robot, its environment, and the actions of the robot. A tiled window approach is used to simplify rapid screen updates. The system supports the display of two different types of windows, 2D and 3D. Each 2D window, used for displaying text and 2D symbolic graphics, has its own associated task responsible for all activities in the

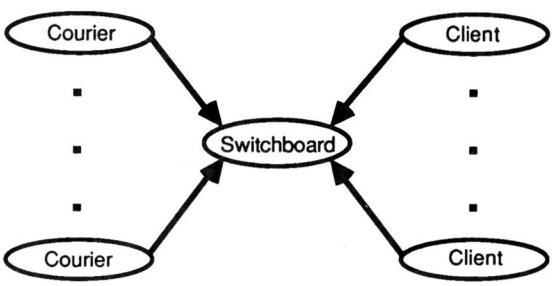


Figure 1. The Switchboard task. (Note that arrow-heads point away from the task making the request.)

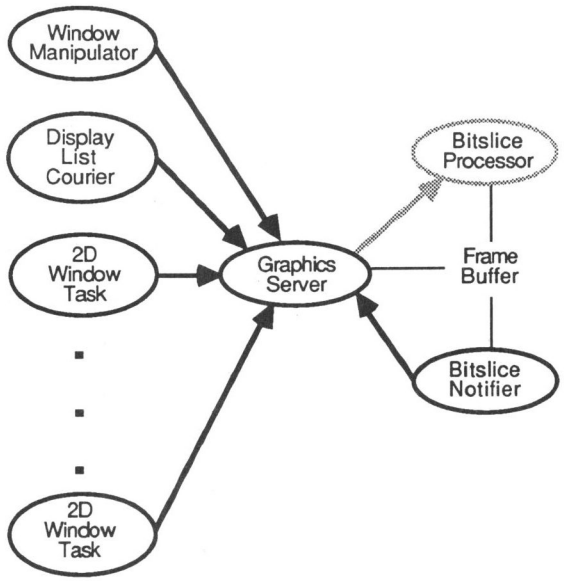


Figure 2. The Graphics Server.

window. All 3D windows, used for displaying different views of the robot and its environment, are controlled by a single task, the Data Structure Server, through the Display List Courier, because each window is a different representation of the same robot data structure.

The Graphics Server is responsible for displaying the contents of both types of windows. As well, it must change the way in which the windows are displayed on the screen in response to window manipulation messages from the Window Manipulator task that request the creation and modification of the window structures maintained by the Graphics Server.

### 2D Graphics

The 2D graphics messages request text or symbolic output to be directed to the screen window associated with the requesting task. The Graphics Server is responsible for translating from the virtual coordinate system of the requestor to the screen coordinates of its window. If required, the messages to a window may be stored and then reinterpreted if a window is modified in size, or if a picture segment that potentially blocks part of the window is moved. This is different from many other tiled window systems that require the application running in the window to become involved with the redraw of a window that has been changed in size.

### 3D Graphics

The 3D graphics messages are in the form of commands for the Graphics Server to modify the display list. The display list is in turn interpreted by the Adage bitslice processor to render the 3D image into the frame buffer [Loo86]. While the microcode running in the bitslice processor is not strictly speaking a Harmony task, and shared memory is used for communication, rather than the send-receive-reply message passing primitives, it can still be viewed externally like any other independent Adagio task. The Graphics Server acts as an *agent* task, as described in Plebon and Booth [Pleb82], for the task running in the bitslice processor, providing an interface to the other processor and managing the communication between and shared resources of the two processors. Because requests for the bitslice processor to update the display list are buffered by the Graphics Server until the bitslice is ready to begin another update, client tasks requesting the services of the Graphics Server do not remain blocked for long.

Currently, the bitslice processor cannot interrupt the 68000 to signal completion of a screen update, so a worker task, the Bitslice Notifier, is created by the Graphics Server with the sole responsibility of informing the Graphics Server when the task running in the bitslice processor has finished. It sleeps most of the time, occasionally waking to poll a flag and sending a notification message to the Graphics Server when the screen update is finished. The incoming requests to modify the display list that the Graphics Server had been buffering are then satisfied, the bitslice processor is released to update the display, and the Graphics Server replies to the Bitslice Notifier thus releasing it to run again.

The display list supports multiple views of a single environment where these views may differ in their viewpoints as well as their display parameters. Modifying display parameters from one 3D window to another permits, for example, the posting of shaded images of the robot in one window and a simple stick figure of only the axes of rotation of each of the joints in another - both rendered from the same display list. Messages, resulting from user actions or simulation of robot activity, can request the rotation or translation of robot links. These require only a change of the appropriate transformation matricies in the display list and a signal to the bitslice processor to re-render the image.

### Data Structure Server

With the multitasking approach to the workstation, it is quite feasible that more than one task would wish to update or query the structure representing the 3D geometry of the robot and its environment. To prevent corruption of the data by concurrent accesses, the data structure is known only to a single task, the Data Structure Server, shown in Figure 3. All requests for information from the data structure, for data structure updates, and for manipulation of 3D windows are fielded by the Data Structure Server. Any update that requires a modification of the screen image is forwarded by the Data Structure Server to the Graphics Server, through the Display List Courier to avoid blocking for a long time. When a major portion of the graphics data structure must be sent to the Graphics Server for the creation of the display list, a pointer to the structure is sent, thus making the structure temporarily known to another task. However, until the Display List Courier reports the completion of the screen update, the Data Structure Server will satisfy only 3D data information requests. Requests to change the data will be held until it is safe to do so.

### Tracker Server

Management of user feedback is an important element in any interactive system. The user must often keep in touch with several activities simultaneously. He must know what actions are available to him and what the system is doing, and he needs reassurances that all is progressing as it should.

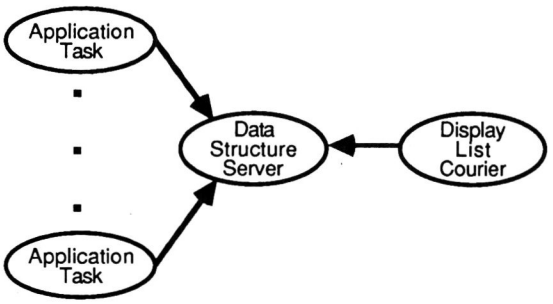


Figure 3. The Data Structure Server.

One element of system feedback is the *screen cursor* or *tracker*. Always displayed at or near the user's centre of attention, a well designed tracker that changes in response to system activity is a powerful indicator of the state of the world. Tilbrook [Tilb76], [Baec80] shows that a tracker can provide much more information than simply the X, Y position of a locator device. Plebon and Booth [Pleb82, pp. 26-28] provides additional information on the use of trackers as feedback and gives further references.

The term *tracker* is used for describing the feedback showing the current X, Y position of a locator device (mouse, tablet, joystick, etc.), because it is simple, it correctly describes the activity (tracking a locator device), and it avoids ambiguity. Although *cursor* is the most widely used term for locator feedback, it also has been associated with the flashing bar, line or box found on most alpha-numeric terminals to prompt for text input, with the hand-held physical device used for positioning on a graphics tablet (puck), and even with the the cross-hair wires at the tip of the puck.

### The Locator Model

The Tracker Server assumes a locator model such as the one provided by the Adagio Tablet Server [Tann85b]. In addition to X, Y coordinates, the Tablet Server returns a *window*, or ID of a predefined region on the tablet surface; and a *status* of the pointing device. The status is dependent on both the current state of the tablet and the state during the previous read operation, and may be one of: UP/UP, UP/NEAR, NEAR/NEAR, NEAR/DOWN, DOWN/DOWN, DOWN/NEAR, or NEAR/UP.

### Icons

In an environment where many activities may be controlled by a single input device, in particular a window based system, the tracker must be able to provide feedback indicating the action being performed. Having the tracker displayed as one of several possible graphical *icons*, or pictorial symbols, can help accomplish this. These icons, drawn from a set of icons defined for the system, (not all of which are used for trackers), enrich the interaction because different ones may be used to indicate the window where the tracker currently resides, the state of the task for that window, and perhaps the button most recently pushed. Figure 4 shows some currently used icons.

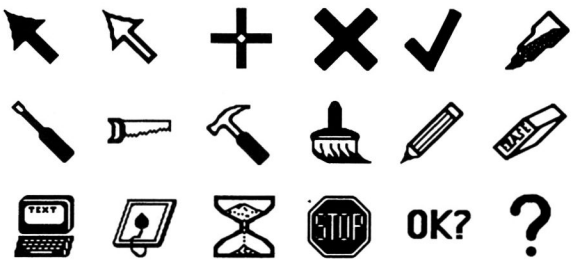


Figure 4. Some Adagio icons.

### The Server Task

The Adagio Tracker Server, shown in Figure 5, offers advantages over some other approaches to tracker management. For example, the University of Waterloo Paint program [Pleb82] [Beac82] also uses a single tracking task, but it is a small worker task, created whenever interaction is permitted and destroyed when it is not. Paint provides little control over icons, they are compiled into the program. The worker task has only three kinds of trackers and relies on a few global variables for information about the trackers.

The Tracker Server, taking advantage of the powerful Harmony server model, maintains the data structure of all icons in Adagio, allowing icons to be *designated* or added, to be removed, or to be modified. Any icon may be *invoked* or made known to the graphics hardware as the current tracker. The Tracker Server can *bind* an icon with a particular screen window, the status of a locator device and the button value of the locator (together called an *icon bundle*).

Icon bundles allow different trackers to be used for different states of the system so that the tracker best reflects the user's current activity. Bundles can be stacked, so that old bundles can be remembered if a temporary action needs to use the same window, status and button information as a previous action. This temporary action could be, for example, the changing of the tracker icon to Tilbrook's Buddha or Macintosh's wristwatch indicating that the window is busy, or when, through some mode selection, a different tracker is required to indicate the new mode. Bundles subsequently can be removed to restore the previous bundle or to break the icon-window-status-button association.

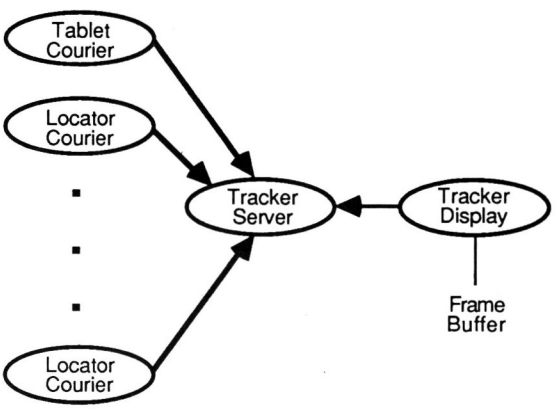


Figure 5. The Tracker Server.

The Tracker Server allows trackers to be positioned in two ways, either by bundle or by naming the icon. Also, the server allows the tracker to be turned on or off as needed. The Tracker Server is device independent, it relies on the device dependent Tracker Display worker task to position the tracker on the screen.

### Complete System

Figure 6 shows many of the tasks discussed in this paper. (Currently one may have 40 to 50 tasks concurrently active.) It is obvious from this figure that rapid message passing is crucial if the user is to see the results of his actions reflected in the image on the screen in a reasonable amount of time. Harmony does indeed provide such a basis, and its existance has encouraged the experimental techniques developed in this project.

### Conclusion

The paper has described a number of servers currently in use in Adagio, each responsible for a certain resource. The use of these servers offers an abstraction between the specific features of the resource and the users or clients of the resource. The Graphics Server offers device independence as do many available graphics subroutine packages. However the Data Structure Server extends the idea of independence into the realm of data structures. The Tracker Server hides from its clients the peculiarities of the particular hardware tracker.

A second advantage of the server approach, coupled with the use of clients, couriers, and notifiers, is the degree to which it makes it simple and natural for people to structure and code programs for multiple tasks and multiple processors. Resulting programs exhibit a high degree of parallelism, making possible the efficient use of multiprocessors - a necessity considering the direction of hardware development.

A high degree of modularity also results from the use of servers. The server model encourages the building of tools that are almost completely self-contained. All details concerning a resource can be easily encapsulated in a single unit. The interface is usually as simple as sending one of a predefined set of messages to the server and expecting one of a small set of predefined replies in return.

Multitasking models of programming for interactive systems are not only useful for reasons of computing efficiency, but provide a far more appropriate base for computer-human interaction. Contrary to the belief of many system designers [Kern84], a human is not a file to be read. The narrow "user is a file" belief has led to the traditional interactive dialogue where the human uses a specific device in reaction to the systems commands. A multitasking system, made more simple to program using tools such as servers, can easily give far more control to the user by providing him with a variety of tools waiting to serve him.

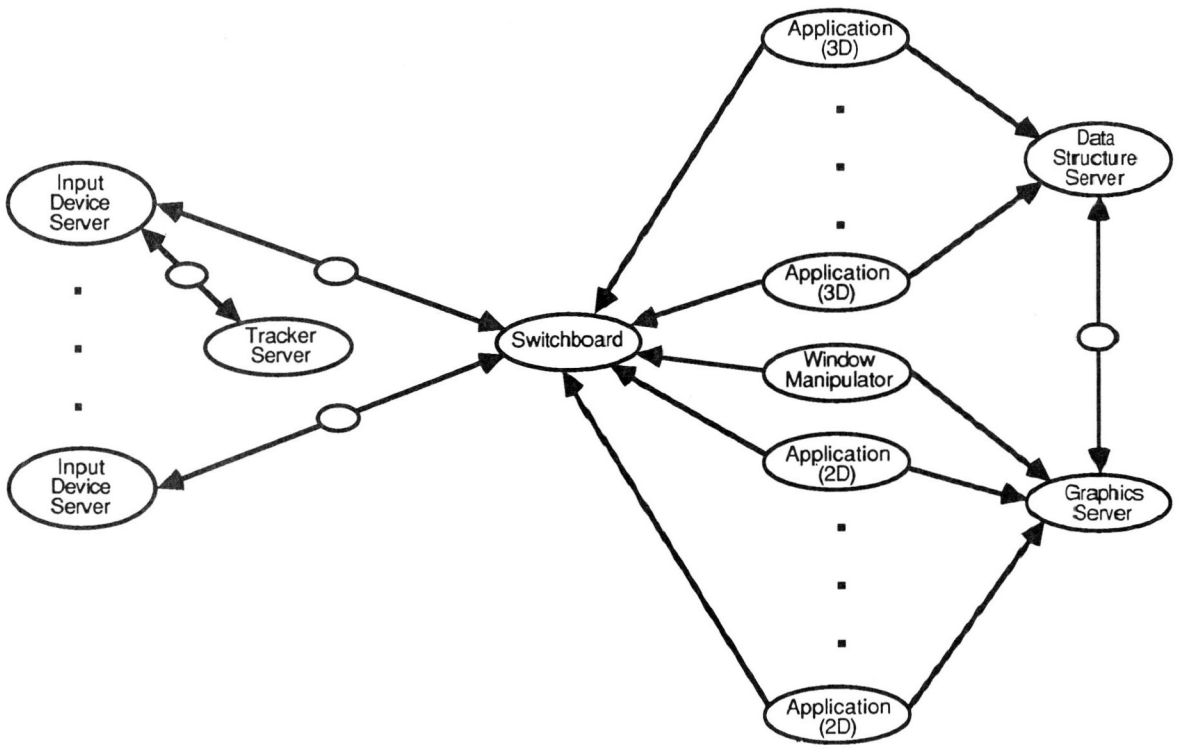


Figure 6. A typical configuration of the complete system. Only major tasks are shown, worker tasks, for example, are omitted. Significant couriers are shown as small unlabelled ovals.

## Bibliography

[Baec80] R.M. Baecker, D.M. Tilbrook, M.I. Tuori, D. McFarland, "Newswhole," SIGGRAPH video review #1, 1980.

[Beac82] R.J. Beach, J.C. Beatty, K.S. Booth, E.L. Fiume, and D.A. Plebon, "The message is the medium: Multiprocess structuring of an interactive paint program," *Computer Graphics,* vol. 16, no. 3, pp. 277-287, July 1982.

[Gent81] W.M. Gentleman, "Message passing between sequential processes: the reply primitive and the administrator concept," *Software Pract. & Exper.*, vol. 11, pp. 436-66, 1981.

[Gent85] W.M. Gentleman, "Using the Harmony operating system," Report of DEE, National Research Council of Canada, NRCC-ERB-966, Ottawa, Ont., Dec. 1983, revised May 1985.

[Kern84] B.W. Kernighan and R. Pike, *The Unix Programming Environment*, Prentice Hall, 1984.

[Loo86] R. Loo, "ARIA - A near-real-time graphics package," M.Math Thesis, Univ. of Waterloo, Dept. of Computer Science, 1986.

[Pleb82] D.A. Plebon and K.S. Booth, "Interactive picture creation systems," Univ. of Waterloo, Dept. of Computer Science, CS-82-46, Dec. 1982.

[Tann85a] P.P. Tanner and M. Wein, "Parallel input in computer-human interaction," *Proc. 18th Annual Meeting, Human Factors Association of Canada,* Hull, Quebec, Sept. 1985.

[Tann85b] P.P. Tanner, M. Wein, W.M. Gentleman, S.A. MacKay, and D.A. Stewart, "The user interface of Adagio, a robotics multitasking multiprocessor workstation," *Proc. 1st International Conference and Exhibition on Computer Workstations,* San Jose, Calif., 1985.

[Tann86] P.P. Tanner, S.A. MacKay, D.A. Stewart, and M. Wein, "A multitasking switchboard approach to user interface management," to appear in *Computer Graphics,* vol. 20, no. 3, August 1986.

[Tilb76] D.M. Tilbrook, "A newspaper pagination system," M.Sc. Thesis, Univ. of Toronto, Dept. of Computer Science, 1976.