Contenido

[1. Caso de estudio del proyecto de Grado 1](#_Toc397095253)

[1.1. Modelado 4](#_Toc397095254)

[1.1.1. Especificación usando MSC 7](#_Toc397095255)

[1.1.2. Diseño usando SDL 23](#_Toc397095256)

[1.2. Pruebas 31](#_Toc397095257)

[1.3. Verificación Formal usando IFx 52](#_Toc397095258)

# Caso de estudio del proyecto de Grado

El sistema de parqueo de la Pontifica Universidad Javeriana Cali (PUJC) ha sido seleccionado como el caso de estudio para realizar en conjunto las verificaciones formales y pruebas planteadas en el presente Trabajo de Grado, con el fin de valorar y minimizar los errores en el diseño de un sistema distribuido.

La Figura 1 muestra el sistema de parqueo de vehículos de la PUJC, el cual cuenta con las siguientes características:

1. Por lo menos con una entrada y una salida principal, para este caso se encuentran ubicadas sobre la avenida Cañasgordas.
2. Cada entrada y salida principal cuenta con un sistema que permite identificar las placas de los carros, un lector de carnés y un sistema básico de sensores que garantiza que el usuario pueda ingresar o salir del sistema de parqueo.
3. Un conjunto de zonas[[1]](#footnote-1) que están bajo el mando de un controlador; en la Figura 1 se muestran 5 controladores del sistema de parqueo de la PUJC que son: Principal, Las Palmas, El Lago, Docentes y Almendros.
4. Repositorios donde se encuentran los usuarios que tienen permitido el acceso al sistema de parqueo y sus placas registradas.

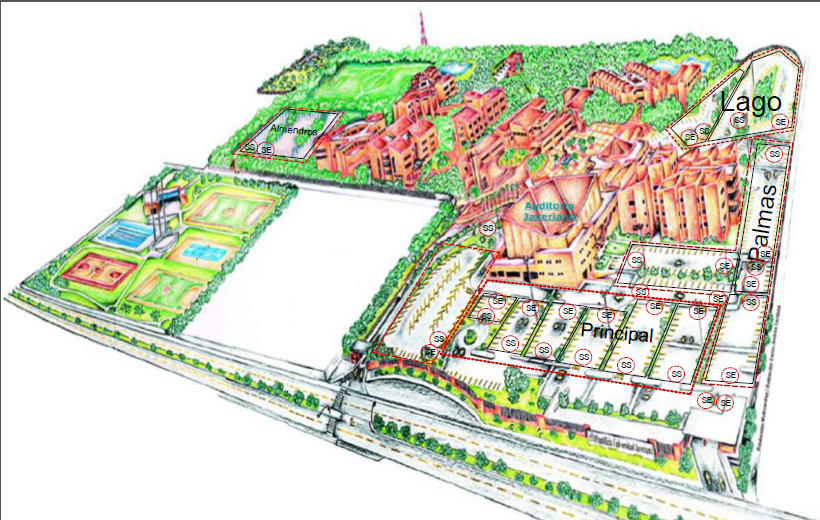


Figura 1 Sistema de Parqueo de la PUJC

La Figura 1 muestra básicamente 5 controladores de zonas que son:

1. Controlador “Principal” que tiene 6 zonas de parqueo.
2. Controlador “Docentes” que tiene 1 zona de parqueo.
3. Controlador “Las Palmas” que tiene 3 zonas de parqueo.
4. Controlador “El Lago” que tiene 2 zonas de parqueo.
5. Controlador “Los Almendros” que tiene 1 zona de parqueos.

La figura 2 ilustra los componentes del sistema de parqueo de la PUJC. Básicamente un sistema de parqueo de vehículos cuenta con *C* controladores de zonas que a su vez pueden tener hasta *Z* zonas; éstas últimas poseen un sistema de sensores capaces de informar a su controlador si un vehículo ingresó o salió de dicha zona. Adicionalmente, el sistema puede tener hasta *E* entradas y *S* salidas principales.

Para garantizar que al sistema sólo ingresen usuarios registrados como (estudiantes, docentes, personal administrativo, colaboradores, etc.), es indispensable contar con elementos que permitan cumplir dicho objetivo. Por ello, el sistema está diseñado para que cada entrada y salida principal cuente con una cámara, cuyo propósito es detectar la placa del vehículo que va a ingresar o salir del sistema. Igualmente, se considera tener un lector de carnés que permite validar si el usuario efectivamente está registrado en el sistema; finalmente se requiere un conjunto de sensores que permitan que el vehículo ingrese o salga del sistema sin ocasionar daño alguno al vehículo.

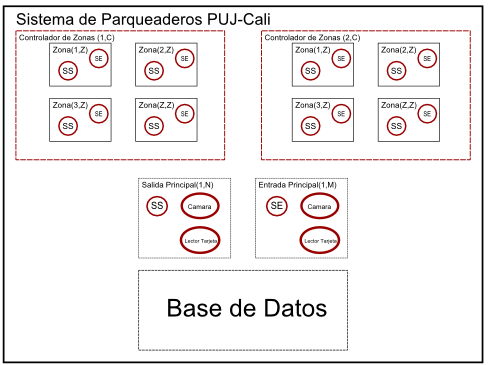


Figura 2. Componentes del Sistema de Parqueo de la PUJC

La Tabla 1 Funciones del sistema de parqueo de la PUJC describe las funciones que posee el sistema de parqueo de la PUJC y el tipo de usuario que puede hacer uso de éstas.

Tabla 1 Funciones del sistema de parqueo de la PUJC

|  |  |  |
| --- | --- | --- |
| **Función** | **Descripción de la función** | **Agente** |
| Crear Controlador de Zona | Permite que el sistema tenga más controladores y a su vez una mayor capacidad de zonas, lo que se traduce en poder aparcar más vehículos. | ADMINISTRADOR |
| Crear una Zona en un Controlador Específico | Permite anexar una zona en un controlador específico ampliando la capacidad de éste último. | ADMINISTRADOR |
| Crear una Entrada o Salida principal | Permite que el sistema tenga más entradas y salidas para el acceso de usuarios al sistema de parqueo. | ADMINISTRADOR |
| Solicitar información a los controladores de zona | Permite saber las plazas libres que posee cada uno de los controladores, lo que posibilita que el usuario sepa en qué controlador y en qué zona hay espacio para aparcar. | ADMINISTRADOR/AUTOGESTIÓN DEL SISTEMA[[2]](#footnote-2) |
| Inicializar el número de plazas totales y libres de una zona de un controlador en específico | Permite inicializar el número de plazas totales y libres de una zona de un controlador en específico, posibilita tener zonas de diferentes capacidades. | ADMINISTRADOR |
| Reportar ingreso o salida de un usuario al sistema de parqueo | Reporta si un usuario ha ingresado al sistema de parqueo si y solo si: El código del usuario y la placa de su vehículo se encuentran registrados en la base de datos del sistema de parqueo. | SISTEMA |
| Reportar el ingreso o salida de un usuario en una zona | Reporta si un vehículo ha ingresado o ha dejado una zona cuando se detecte una secuencia específica en los sensores de la zona. | SISTEMA |

## Modelado

El modelado del comportamiento global del sistema se hizo usando MSCs y el modelo detallado de cada componente del sistema de parqueo se hizo usando el lenguaje SDL, ambos sobre la herramienta RTDS; el modelo del sistema se desarrolló a partir de las funcionalidades descritas en la tabla 1.

La arquitectura del sistema se divide en dos grandes bloques y estos a su vez se dividen en más bloques y procesos. La figura 3 detalla la arquitectura general del sistema de parqueo de la PUJC.

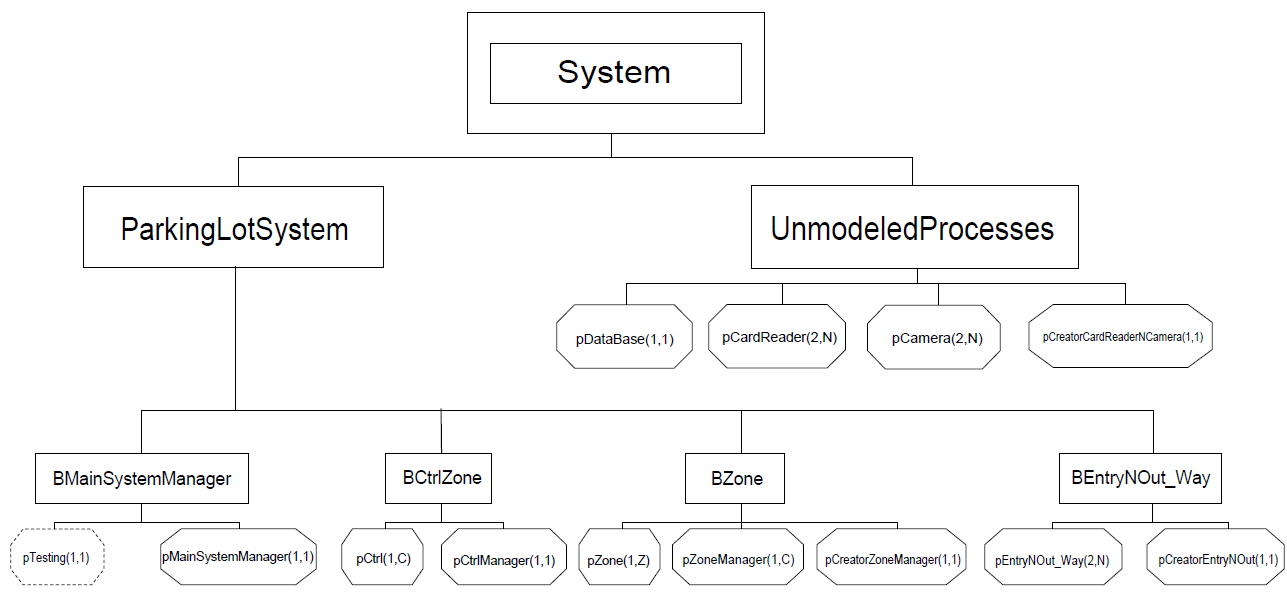


Figura 3. Arquitectura del modelo del sistema de parqueo de la PUJC

Como se aprecia en la figura 3, el sistema cuenta con dos bloques principales llamados ParkingLotSystem y UnmodeledProcesses. El primer bloque está conformado por los procesos y elementos que se han modelado con un mayor nivel de detalle; por el contrario, en el segundo bloque se encuentran los elementos del sistema que no se han modelado sino que se han considerado como procesos, los cuales reciben un valor y retornan otro pero sin realizar un procesamiento específico de la información, dado que no es de interés del proyecto modelarlos.

Las tablas 2 y 3 describen las funciones generales de cada bloque y de los procesos del sistema de parqueo, respectivamente:

Tabla 2. Descripción de los bloques del sistema de parqueo.

|  |  |  |
| --- | --- | --- |
| **Bloques** | **Sub-bloques** | **Propósito/Función** |
| UnmodeledProcesses | - | En este bloque se da lugar a la validación de usuarios permitidos para ingresar o salir del sistema de parqueo. |
| ParkingLotSystem | BMainSystemManager | Este bloque interactúa principalmente con el Administrador, con el propósito de caracterizar el sistema de parqueo. Por ejemplo: Crear controladores de zonas, zonas, entradas y salidas principales, etc. |
| BCtrlZone | Interactúa básicamente con los bloques BMainSystemManager y BZone. Captura la información del conjunto de zonas asociadas a un controlador y las reporta a bloque BMainSystemManager cada vez que haya un requerimiento de información. |
| BZone | Interactúa principalmente con el bloque BCtrlZone y el usuario. Reporta el ingreso y salida de vehículos a partir de la información proveniente de los sensores. |
| BEntryNOut\_Way | Interactúa principalmente con los procesos del bloque UnmodeledProcesses y con el usuario. En éste bloque se da lugar al ingreso y salida del usuario por las entradas y salidas principales, respectivamente. |

Dentro de la arquitectura del sistema de parqueo presentada en la figura 3 se detallan algunos procesos que tienen como función de crear instancias, con el objetivo de hacer que el sistema de parqueo sea escalable de manera sencilla.

En términos del lenguaje, dado que la arquitectura del sistema de parqueo se compone de bloques, no es posible crear dinámicamente instancias de procesos que se encuentren en otro bloque. Lo anterior implica tener procesos que solo sirvan para instanciar otros procesos que se encuentren en su mismo bloque.

|  |  |
| --- | --- |
| **Nombre del proceso** | **Funciones** |
| pDataBase | Este proceso representa las bases de datos del sistema, además está encargado de evaluar si un usuario está habilitado para ingresar o salir del sistema de parqueo. |
| pCardReader | Este proceso representa al lector de carnés y se encarga de entregar el código del usuario que está por ingresa o salir del sistema de parqueo. |
| pCamera | Este proceso representa la cámara que captura la placa del vehículo y la entrega al sistema para evaluar si el usuario está permitido para ingresar o salir del sistema. |
| pCreatorCardReaderNCamera | Proceso encargado de crear los procesos pCardReader y pCamera, y asignarlos a su respectivo proceso pEntryNOut\_Way. |
| pEntryNOut\_Way | Este proceso representa una entrada o salida principal, es el encargado de permitir el paso del usuario al sistema de parqueo. Adicionalmente es el único proceso que interactúa con los procesos pDataBase, pCardReader, pCamera y pCreatorCardReaderNCamera. |
| pCreatorEntryNOut\_Way | Proceso encargado de crear el proceso pEntryNOut\_Way. |
| PZone | Este proceso representa una zona del sistema e interactúa con el controlador de zonas reportando el ingreso o salida de un vehículo, dependiendo de la secuencia de señales del conjunto de sensores que se reciba. |
| pZoneManager | Proceso encargado de crear instancias del proceso pZone y a su vez asignarle su respectivo proceso pCtrl. |
| pCreatorZoneManager | Proceso encargado de crear los procesos pZoneManager y asignarle a éste su respectivo controlador de zonas. |
| pCtrl | Este proceso representa un controlador de zonas y está encargado de controlar los procesos pZone, adicionalmente reporta el estado de las plazas libres y totales que posee cada zona. |
| pCtrlManager | Proceso encargado de crear los procesos pCtrl. |
| pMainSystemManager | Este proceso representa la interfaz directa entre el administrador y el sistema. A través de este proceso se pueden crear entradas y salidas principales, controladores de zona, zonas, hacer requerimientos de información, inicializar tanto plazas libres como totales. |
| pTesting | Proceso que permite acoplar el sistema para hacer pruebas funcionales de caja negra. Este proceso no debe ser considerado en la fase de implementación del sistema de parqueo. |

Tabla 3. Descripción los procesos del sistema de parqueo.

### Especificación usando MSC

El modelado del sistema de parqueo empieza desde los requerimientos del sistema que se describen en la tabla 1. La herramienta RTDS tiene la posibilidad de anexar al proyecto diagramas MSC; éstos son fundamentales para describir las especificaciones del sistema. Existen estrategias para el diseño y desarrollo de modelos, entre las cuales encontramos *Top-Down* y *Bottom-Up*. Top-Down es una estrategia donde se empieza desarrollando el modelo desde un alto nivel con una alta abstracción y termina en bloques o procesos con menores niveles de abstracción. Bottom-Up es una estrategia que consiste en empezar a diseñar los elementos del sistema desde un nivel de abstracción bajo y a partir de éstos se construyen módulos más complejos. La estrategia Top-Down es usada para la especificación del sistema empleando diagramas MSC, y para el diseño del sistema se hace uso de la metodología Botton-Up empleando SDL.

#### MSCs de las especificaciones del sistema

Metodológicamente es recomendable especificar el sistema en alto nivel sin tener en cuenta los detalles de cada módulo; para ello, se elabora un MSC que tiene como finalidad mostrar un escenario en el cual se describa la interacción de los dos bloques principales: ParkingLotSystem y UnmodeledProcesses. La figura 4 muestra dicha interacción; el escenario que se describe es el acceso de un vehículo al sistema de parqueo.

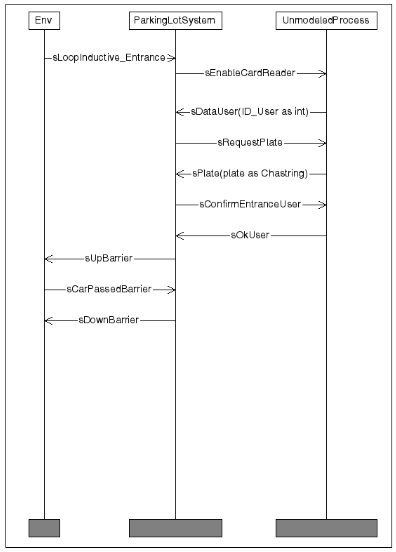


Figura 4. MSC interacción entre los dos bloques ParkingLotSystem y UnmodeledProcesses

En este MSC encontramos tres agentes interactuando entre sí los cuales son: Env, ParkingLotSystem y UnmodeledProcesses. El agente Env representa el ambiente o medio por el cual el sistema intercambia mensajes con el usuario o con los sensores. El procedimiento que describe el acceso de un vehículo al sistema de parqueo de la PUJC está dado por la siguiente secuencia:

1. El proceso Env envía una señal al bloque ParkingLotSystem llamada sLoopInductive\_Entrance que indica que un vehículo está en la portería principal.
2. El bloque ParkingLotSystem necesita verificar que el usuario que desea ingresar al sistema de parqueo está autorizado. Para ello, necesita saber su identificación y la placa del vehículo. Con la señal sEnableCarReader habilita el lector de carnés.
3. Una vez que el bloque ParkingLotSystem tenga el código del usuario, habilita la toma de la foto a la placa, para ello intercambia el mensaje sRequestPlate con el bloque UnmodeledProcesses.
4. El bloque UnmodeledProcesses envía la placa como un charstring al bloque que hizo la consulta.
5. Una vez que se tenga la información del usuario es indispensable validarla y determinar si se le otorga el acceso o no al sistema. Para ello envía un requerimiento al bloque UnmodeledProcesses esperando la señal sOkUser que confirma que el usuario está habilitado para usar el sistema de parqueo o sNoRegis\_User que indica que el usuario no está permitido para usar el sistema de parqueo.
6. Si el usuario está habilitado para usar el sistema de parqueo entonces se efectúa una serie de pasos para darle bienvenida al usuario. Para ello se envía una señal a la talanquera para que permita el paso del vehículo, una vez que éste haya pasado, un sensor envía dicha señal indicando que es seguro bajar la talanquera y el bloque ParkingLotSystem culmina enviando la señal sDownBarrier, quedando así disponibles los dos bloques para interactuar nuevamente.

La anterior secuencia representa el comportamiento actual del sistema de parqueo de la PUJC. Una posible mejora a éste consistiría que la cámara capturara el valor de la placa del vehículo cuando haya presencia de éste, lo que permitiría agilizar el proceso de validación del usuario.

**Comportamiento cuando sale un vehículo.** El escenario que representa la salida de un vehículo del sistema de parqueo de la PUJC es similar al comportamiento descrito en la figura 4. La diferencia radica en que el agente Env o entorno envía una señal sLoopInductive\_Exit en lugar de sLoopInductive\_Entrance cuando hay presencia de vehículo, y en el momento que el bloque ParkingLotSystem desea verificar si el usuario está autorizado para salir, envía una señal sConfirmOutUser.

**Definición de parámetros iniciales del sistema** Los MSCs son útiles para describir la interacción entre agentes, que si bien pueden ser bloques como en el caso previo, también sirven para describir el comportamiento a nivel de procesos. Esto se ilustra en las figuras 5 y 6 que representan los MSCs de la inicialización del sistema cuando el administrador define los parámetros iniciales del sistema de parqueo.

La figura 5, detalla la interacción entre procesos del bloque ParkingLotSystem y los procesos del bloque UnmodeledProcesses. El sistema debe de tener por lo menos una entrada y una salida principal.

* Cada entrada y salida principal debe contar con una cámara la cual retorna el valor de la placa del vehículo a partir de una foto y un lector de carnés.
* El sistema tiene por lo menos una zona con su respectivo controlador. La zona cuenta con sus respectivos sensores para indicar si un vehículo ha ingresado o salido de dicha zona.
* Cuenta con un repositorio de datos donde se encuentran los códigos y sus placas asociadas que están autorizadas para el acceso o salida del sistema de parqueo de la PUJC.

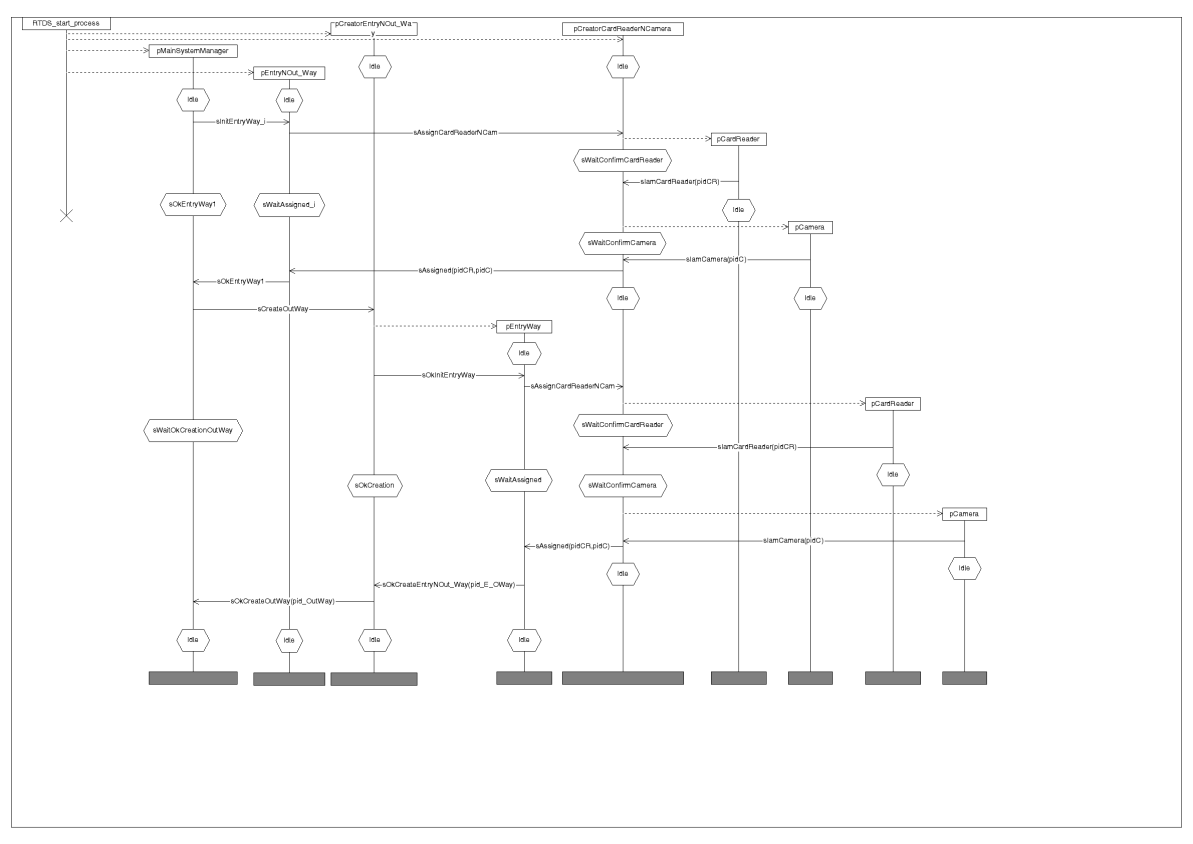


Figura 5. Inicialización entrada y salida principal del sistema

Inicialmente el sistema contiene una instancia de cada uno de los procesos de la figura 3. El proceso pMainSystemManager se encarga de asociar el proceso pEntryNOut\_Way a su tabla de entradas principales; para ello envía una señal llamada sInitEntryWay\_i al proceso pEntryNOut\_Way. A partir de esta solicitud lo que va a pasar es que éste proceso necesita que se le sea asignado un lector de carnés, pCardReader, y una cámara, pCamera. El proceso que tiene la facultad de instanciar estos procesos es pCreatorCardReaderNCamera, una vez éste haya creado el lector de carnés y la cámara, enviará una señal al proceso pEntryNOut\_Way con los identificadores correspondientes. El proceso pMainSystemManager cuando reciba la señal sOkEntryWay1 por parte de pEntryNOut\_Way indicará que la primera entrada principal ha sido exitosamente creada, entonces se procede a crear una salida principal. La interacción de los procesos es muy parecida a la inicialización de la primera entrada principal, solo cambia que el proceso pMainSystemManager solicita al agente pCreatorCardReaderNCamera instanciar un proceso pEntryNOut\_Way y envíe la señal correspondiente a este último para que se le sea asociado un lector de carnés y una cámara. El proceso de inicialización termina cuando el proceso de creación de entradas y salidas principales envía un mensaje al proceso pMainSystemManager llamada sOkCreateOutWay asociando el identificador de dicha salida principal para que sea anexado en su lista de salidas principales.

La figura 6 describe la inicialización de los procesos del bloque ParkingLotSystem, el objetivo de esta interacción es asociar a la zona y al proceso pZoneManager su respectivo controlador. El sistema de parqueo cuenta con controladores de tienen a su mando un conjunto de zonas; dado que el sistema es dinámico es posible que un controlador pueda tener más zonas, ya que el proceso pCtrl no se encuentra en el mismo bloque que pZone se necesita un proceso que sea capaz de crear más zonas, para ello se requiere del proceso pZoneManager. Lo anterior implica que cada proceso pCtrl tendrá asociado hasta *Z* zonas y un único proceso pZoneManager.

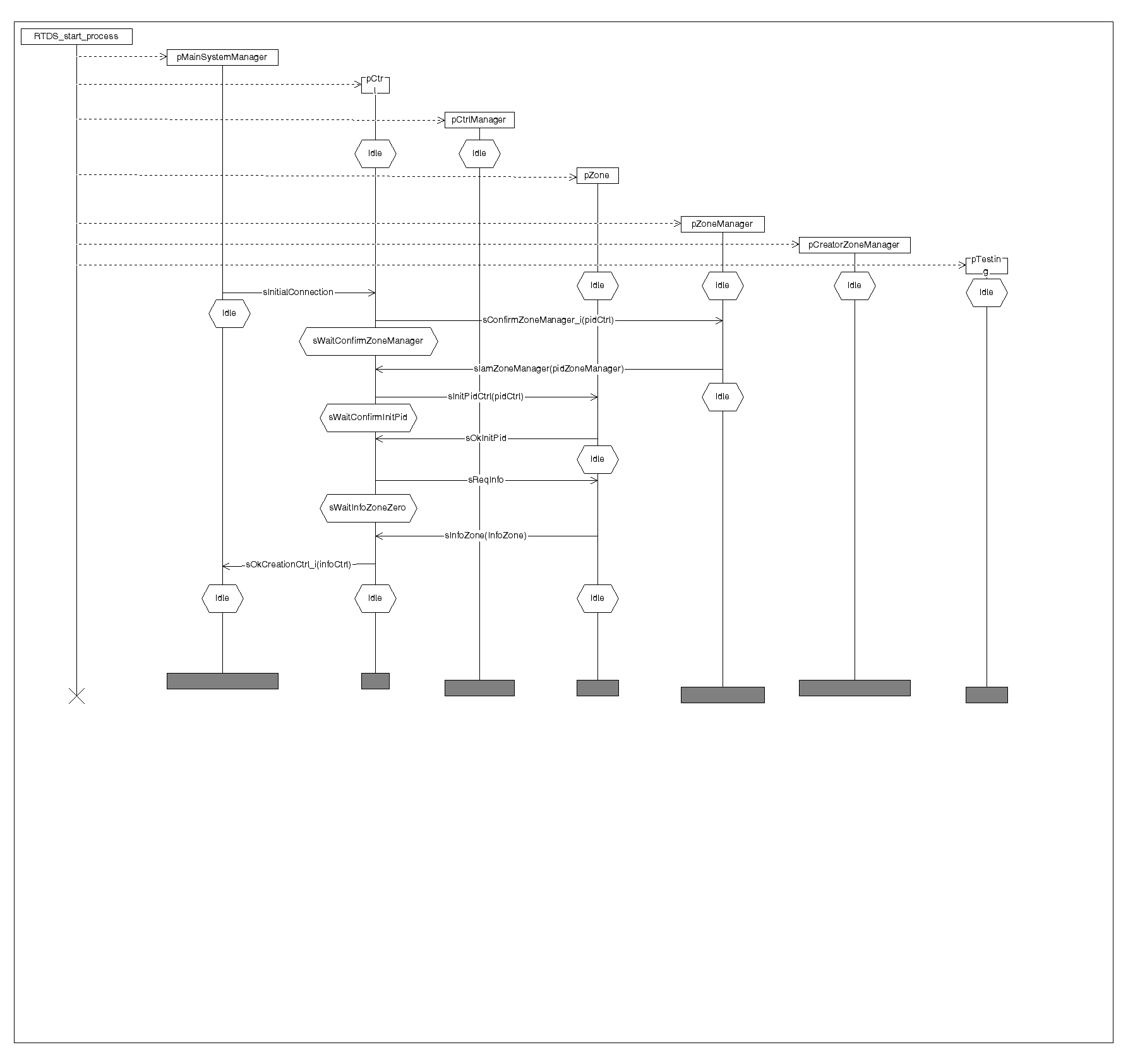


Figura 6. Inicialización de procesos del bloque ParkingLotSystem

La función de cada uno de las señales presentadas a partir de este momento se puede detallar en el Anexo B.

La tabla 4 explica el propósito de cada uno de los MSCs que se diseñaron para la descripción de las especificaciones del sistema de parqueo de la PUJC.

|  |  |
| --- | --- |
| **MSC** | **Descripción** |
| Figura 7. Creación de una entrada principal al sistema de parqueo. | El administrador solicita la creación de una entrada principal al sistema de parqueo. Básicamente se crea el proceso que representa una entrada principal y a éste se le asigna su respectiva cámara y lector de carné. Si el sistema ha creado exitosamente la entrada principal, retorna una señal al administrador llamada sOkCreateE\_W. Si el sistema tiene el cupo máximo de entradas principales retornará al administrador la señal sExcEntryWay en lugar de sOkCreateE\_W y no se efectuará el intercambio de señales para la asignación de cámara y lector de carnés. |
| Figura 8. Creación de una salida principal al sistema de parqueo | El administrador solicita la creación de una entrada principal al sistema de parqueo. Básicamente se crea el proceso que representa una entrada principal y a éste se le asigna su respectiva cámara y lector de carné. El sistema si ha creado exitosamente la entrada principal, retorna una señal al administrador llamada sOkCreateE\_O. Si el sistema tiene el cupo máximo de entradas principales retornará al administrador la señal sExcOutWay en lugar de sOkCreateE\_O y no se efectuará el intercambio de señales para la asignación de cámara y lector de carnés. |
| Figura 9. Creación de una zona de parqueo cuando el controlador de zonas tiene asociado un pZoneManager | El administrador del sistema envía una señal al proceso pMainSystemManager para crear una zona en un controlador específico, con una inicialización de plazas libres y totales de la zona. Para la solicitud de la zona se usa la señal sAddZone donde su primer parámetro es el número del controlador de zona, el segundo y tercero son las plazas totales y libres de la zona respectivamente. Si la creación de la zona es exitosa el proceso pMainSystemManager lo hará saber al administrador a través de la señal sOkCreateZone de lo contrario enviará la señal sExcLimitZones, ésta señal se da antes que el proceso pZoneManager haga una instancia del proceso pZone. |
| Figura 10. Creación de una zona de parqueo cuando el controlador de zonas no tiene asociado un pZoneManager | El administrador del sistema envía una señal al proceso pMainSystemManager para crear una zona en un controlador específico. Este escenario es muy parecido al de la figura 9 la diferencia es que el controlador de zonas no tiene un proceso pZoneManager asociado, por lo cual solicita su creación, una vez creado el pZoneManager y asociado a su respectivo controlador de zonas el intercambio de señales respecto a la figura 9 son los mismos. |
| Figura 11. MSC ajustes de parámetros a una zona de parqueo recién creada. | Este MSC representa el intercambio de señales para la inicialización de una zona de parqueo, en este diagrama a la zona se le asigna un controlador y se ajusta su capacidad de aparcamiento de vehículos. |
| Figura 12. MSC creación de un controlador de zonas al sistema de parqueo | El administrador solicita al sistema de parqueo crear un controlador de zonas por medio de la señal sCreateCtrl que recibe el proceso pMainSystemManager. Si la creación es exitosa el proceso pMainSystemManager retornará una señal sOkCreateCtrl al administrador, de lo contrario, hará saber al administrador que no es posible tener más controladores de zonas con la señal sExcLimitCtrl. |
| Figura 13. MSC ingreso de un vehículo al sistema de parqueo. | Este escenario es similar al de la figura 4 presentada anteriormente pero a nivel de procesos. Dado que los procesos pCardReader, pCamera y pDataBase no han sido modelados, se ha colocado un temporizar en la vida del proceso pCamera que se llama timerProcessOCR el cual representa el tiempo de cómputo que tardaría el proceso pCamera para entregar a partir de una foto la placa de un vehículo en un tipo de dato charstring. |
| Figura 14. MSC Salida de un vehículo del sistema de parqueo. | Este escenario es similar al de la figura 13. La diferencia es que en este escenario el vehículo está por salir del sistema de parqueo, básicamente el proceso de verificación de usuario habilitado para salir es el mismo que cuando el usuario está por ingresar con la diferencia que ésta validación se hace a través de la señal sConfirmOutUser. |
| Figura 15. MSC usuario no autorizado intentando ingresar al sistema de parqueo de la | Este escenario representa el caso cuando un usuario desea ingresar al sistema de parqueo de la PUJC pero éste no está habilitado para hacerlo. La dinámica de verificación del usuario es la misma que la figura 13, solo que la respuesta por parte del proceso pDataBase es negativa y no se le da acceso al usuario de ingresar. |
| Figura 16. MSC usuario no autorizado intentando salir del sistema de parqueo de la PUJC. | Este escenario representa el caso cuando un usuario desea salir del sistema de parqueo de la PUJC pero éste no está habilitado para hacerlo. La dinámica de verificación del usuario es la misma que la figura 14, solo que la respuesta por parte del proceso pDataBase es negativa y no se le da acceso al usuario de salir. |
| Figura 17. MSC usuario ingresando a una zona del sistema de parqueo. | Este MSC representa el escenario el cual un vehículo va a ingresar a una zona del sistema. Para saber que un usuario ha ingresado se tiene que cumplir la siguiente secuencia: Se interrumpe el primero sensor infrarrojo, sIR1\_Zone, posteriormente el vehículo interrumpe el sensor infrarrojo 2, sIR2\_Zone y finalmente se verifica que es un vehículo por la señal capturada en el sensor inductivo sLoopInductive\_Zone. La zona reportará que ha ingresado un vehículo a su respectivo controlador con la señal sEntered\_Car. Otra posibilidad que es válida para el ingreso del vehículo es cuando se interrumpe el sensor infrarrojo 4, luego el sensor infrarrojo 3 y finalmente la recepción de la señal del loop inductivo. |
| Figura 18. MSC usuario saliendo de una zona del sistema de parqueo. | Este MSC representa el escenario el cual un vehículo va a salir de una zona del sistema. Para saber que un usuario ha ingresado se tiene que cumplir la siguiente secuencia: Se interrumpe el primero sensor infrarrojo 3, sIR3\_Zone, posteriormente el vehículo interrumpe el sensor infrarrojo 4, sIR4\_Zone y finalmente se verifica que es un vehículo por la señal capturada en el sensor inductivo sLoopInductive\_Zone. La zona reportará que ha salido un vehículo de su zona a su respectivo controlador con la señal sOut\_Car. Otra posibilidad que es válida para reconocer que un vehículo va a salir de la zona es cuando se interrumpe el sensor infrarrojo 2, luego el sensor infrarrojo 1 y finalmente la recepción de la señal del loop inductivo. |
| Figura 19. MSC Ingreso y salida de un vehículo a través del proceso Testing | Este MSC representa la entrada y salida de vehículos en una zona del sistema, imagen lado izquierdo y derecho respectivamente. La diferencia de este escenario a los de la figura 17 y 18 es que las señales de los sensores son simuladas desde el proceso pMainSystemManager. El controlador y la zona que se quiere entrar y sacar vehículos se da a través del proceso temporal pTesting. Cabe destacar que las señales de sensores generadas por el proceso pMainSystemManager no deben ser considerados en la implementación del sistema de parqueo. |

Tabla 4. Descripción de los MSC correspondientes a la especificación del sistema.

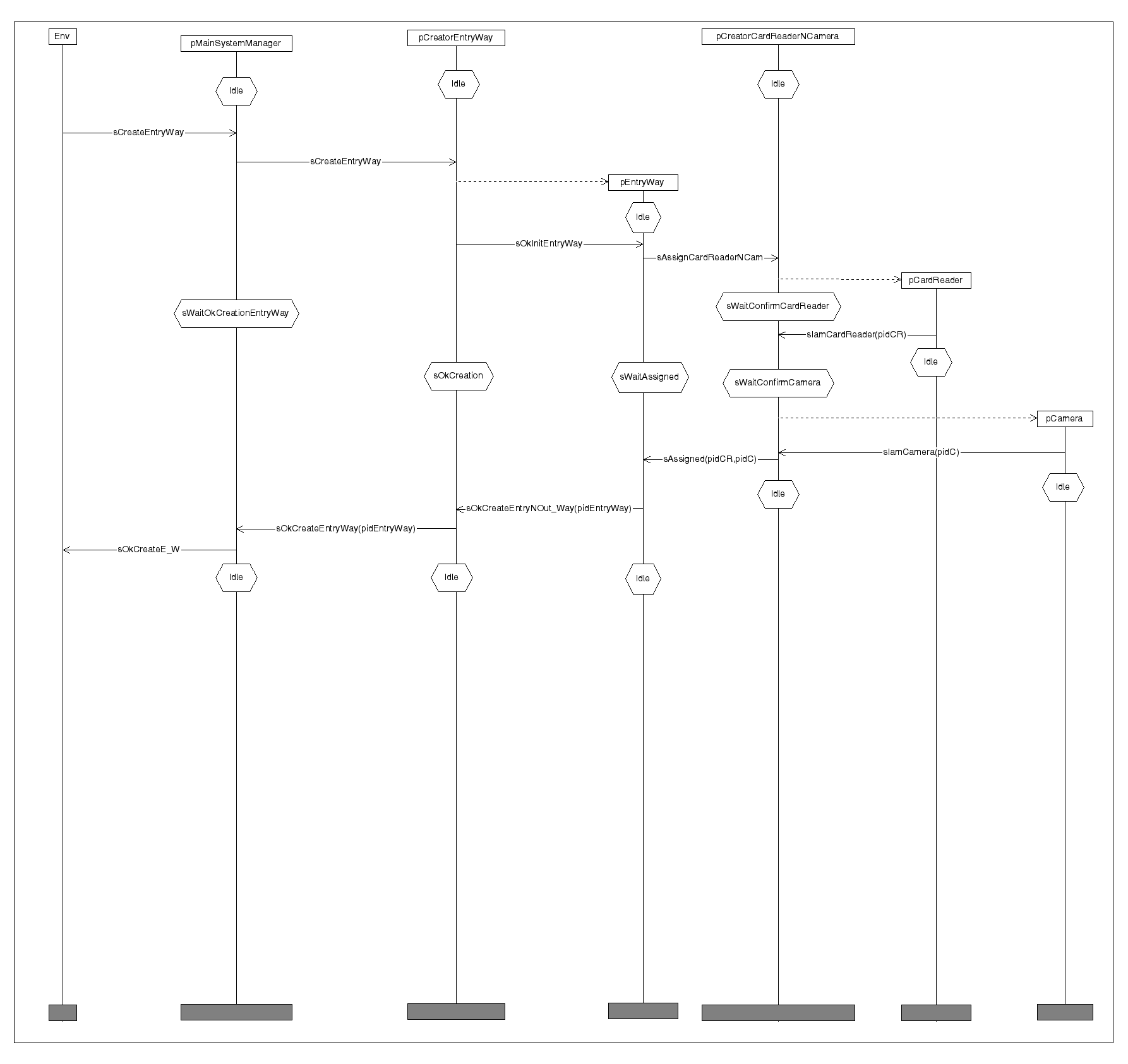


Figura 7. Creación de una entrada principal al sistema de parqueo.

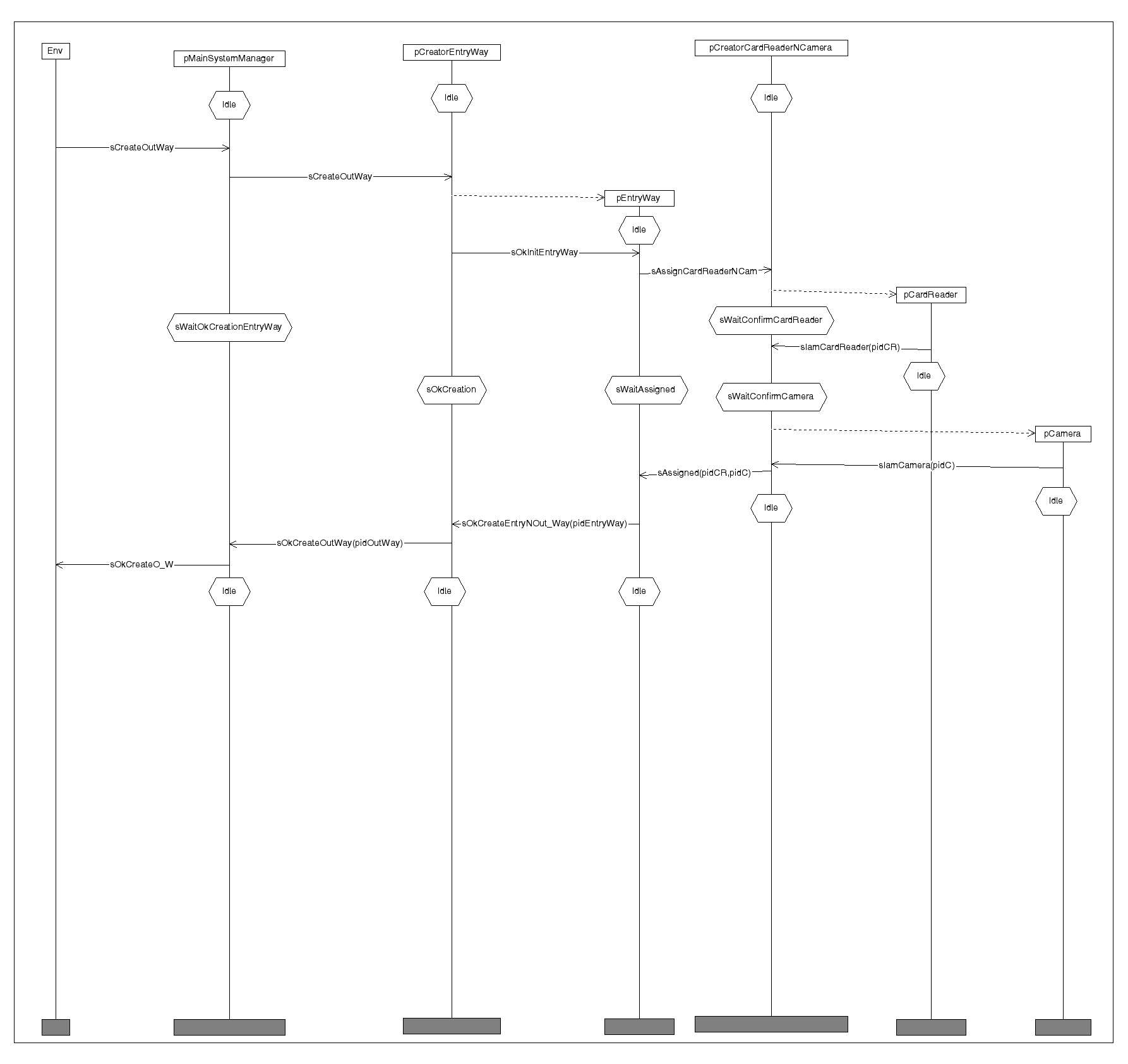


Figura 8. Creación de una salida principal al sistema de parqueo

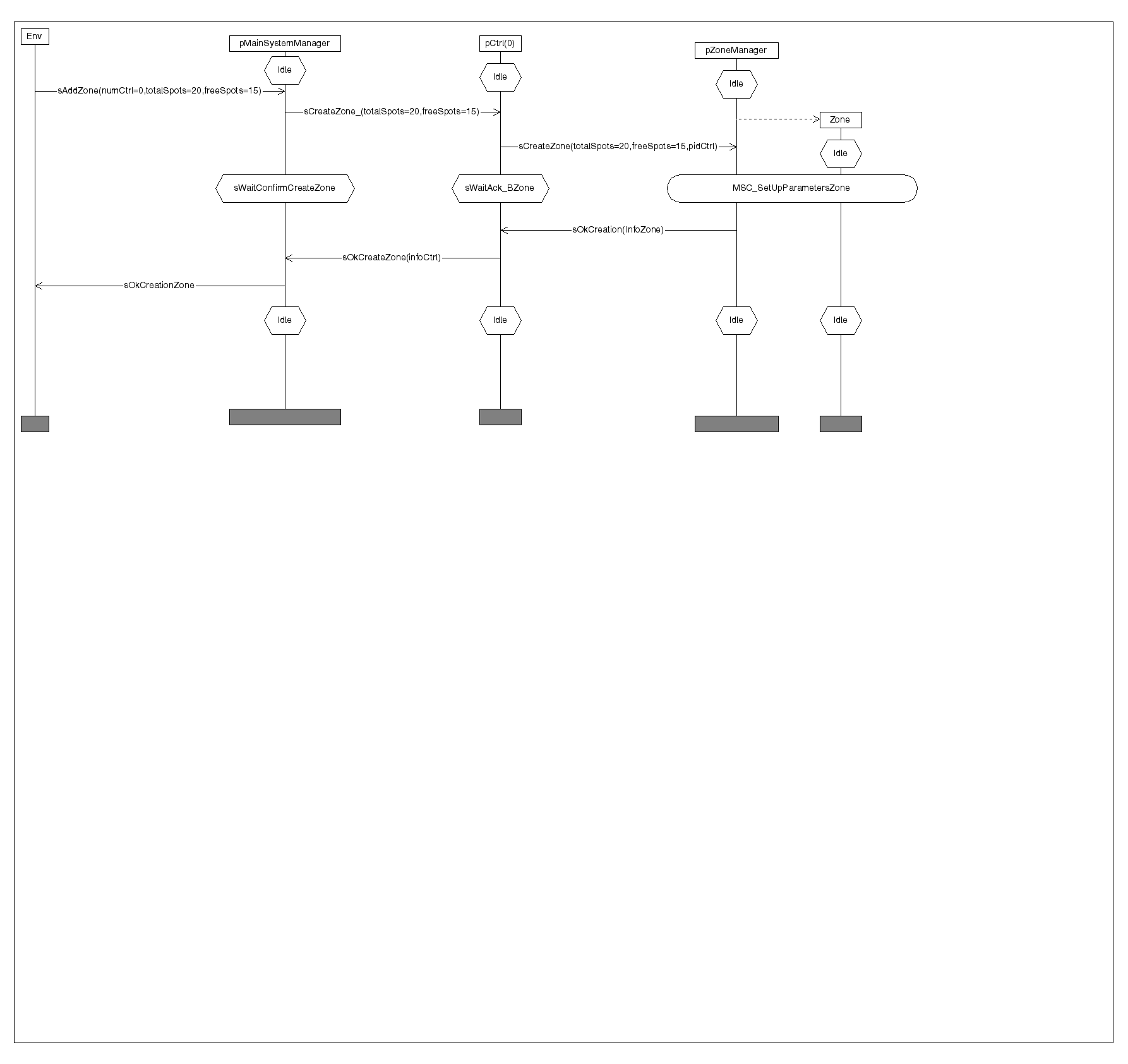


Figura 9. Creación de una zona de parqueo cuando el controlador de zonas tiene asociado un pZoneManager

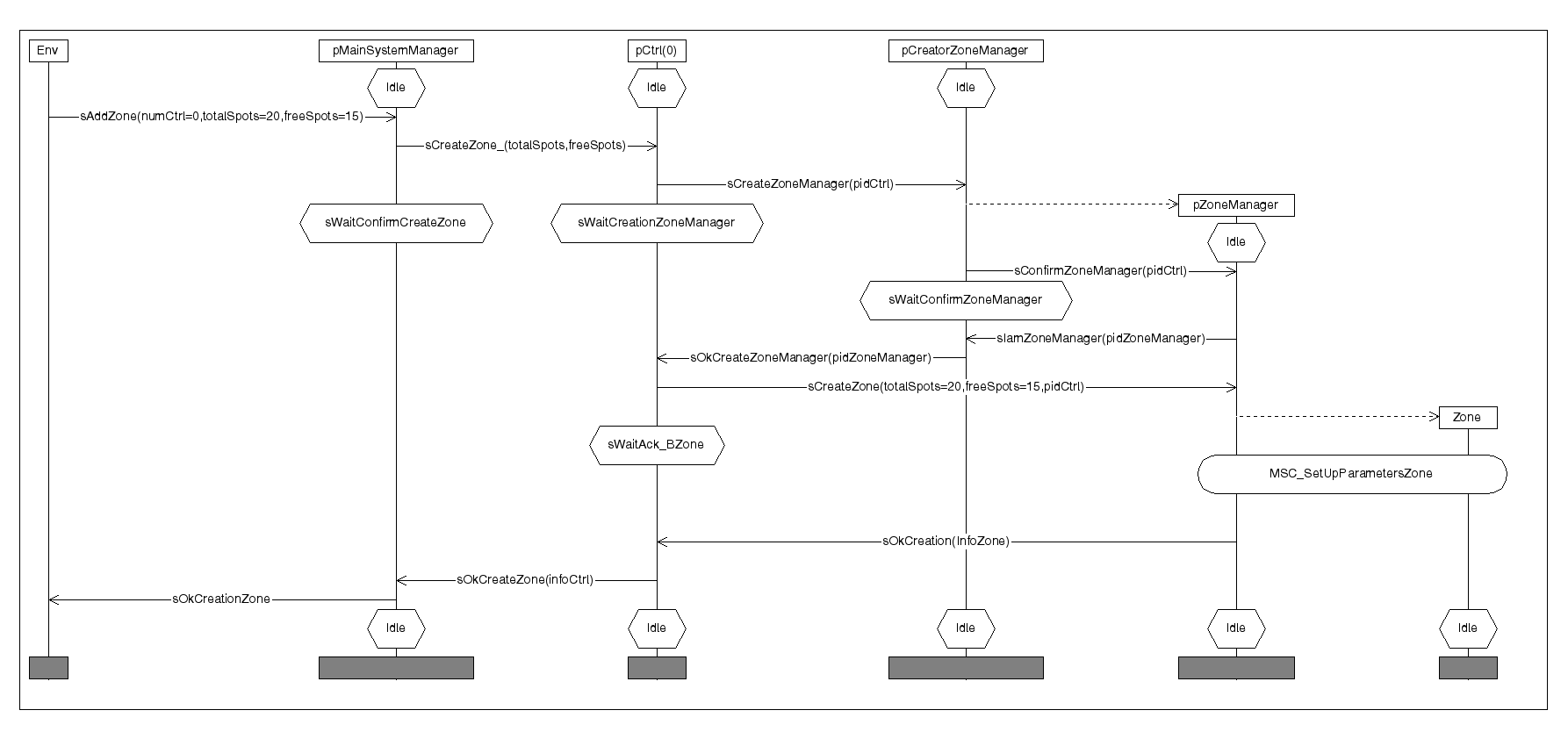


Figura 10. Creación de una zona de parqueo cuando el controlador de zonas no tiene asociado un pZoneManager

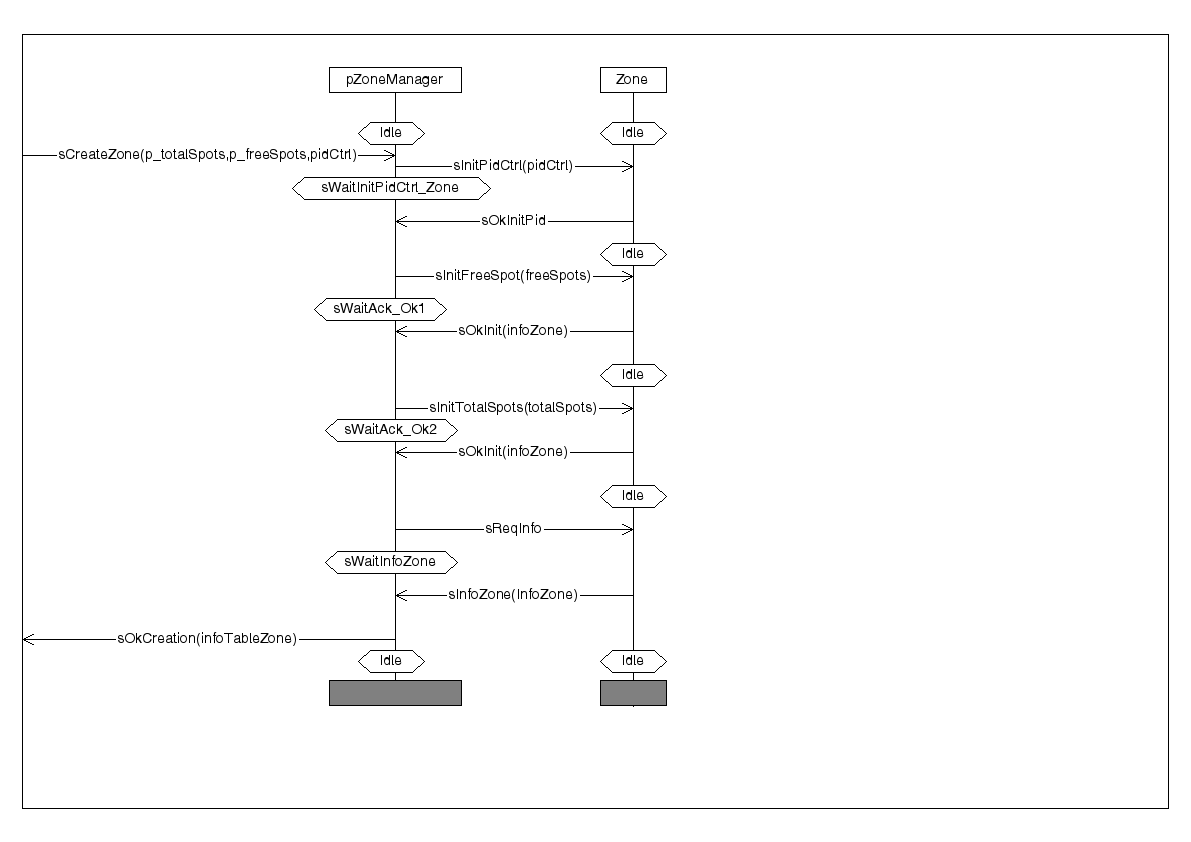


Figura 11. MSC ajustes de parámetros a una zona de parqueo recién creada.

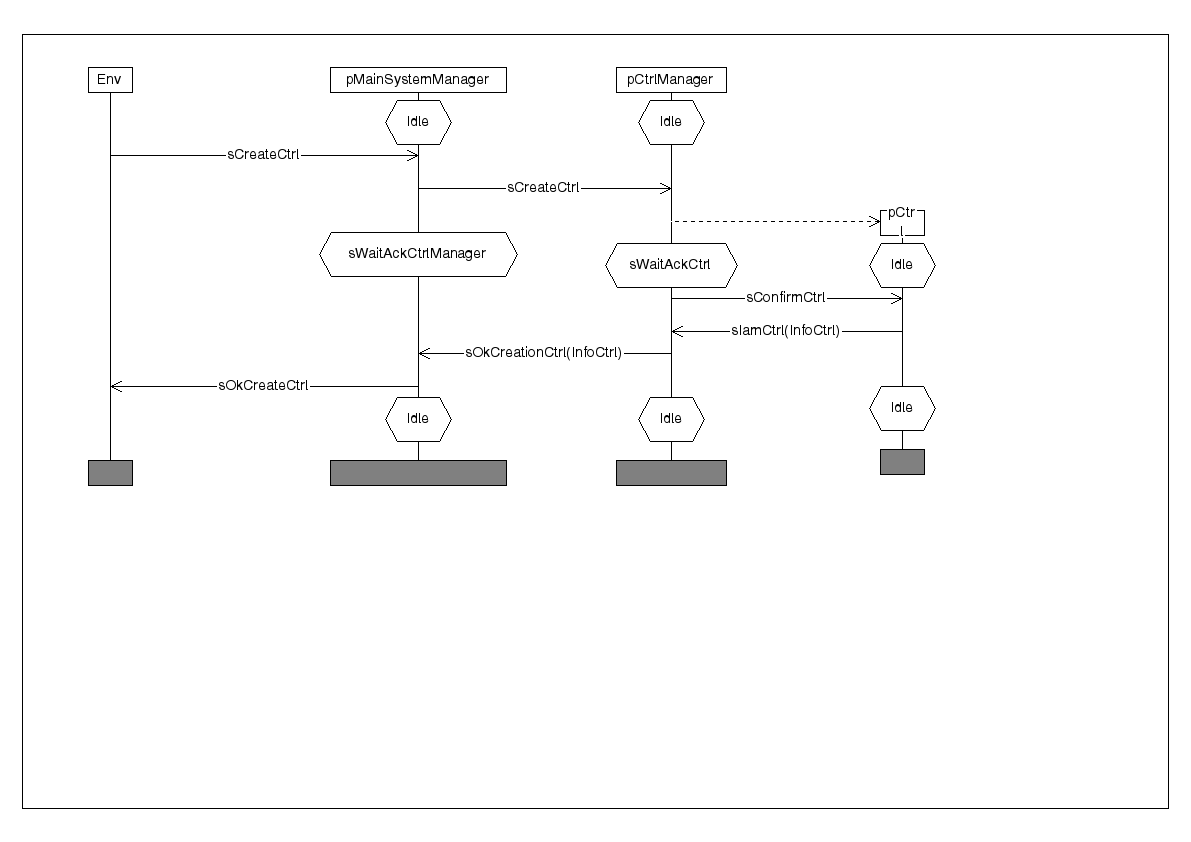


Figura 12. MSC creación de un controlador de zonas al sistema de parqueo.

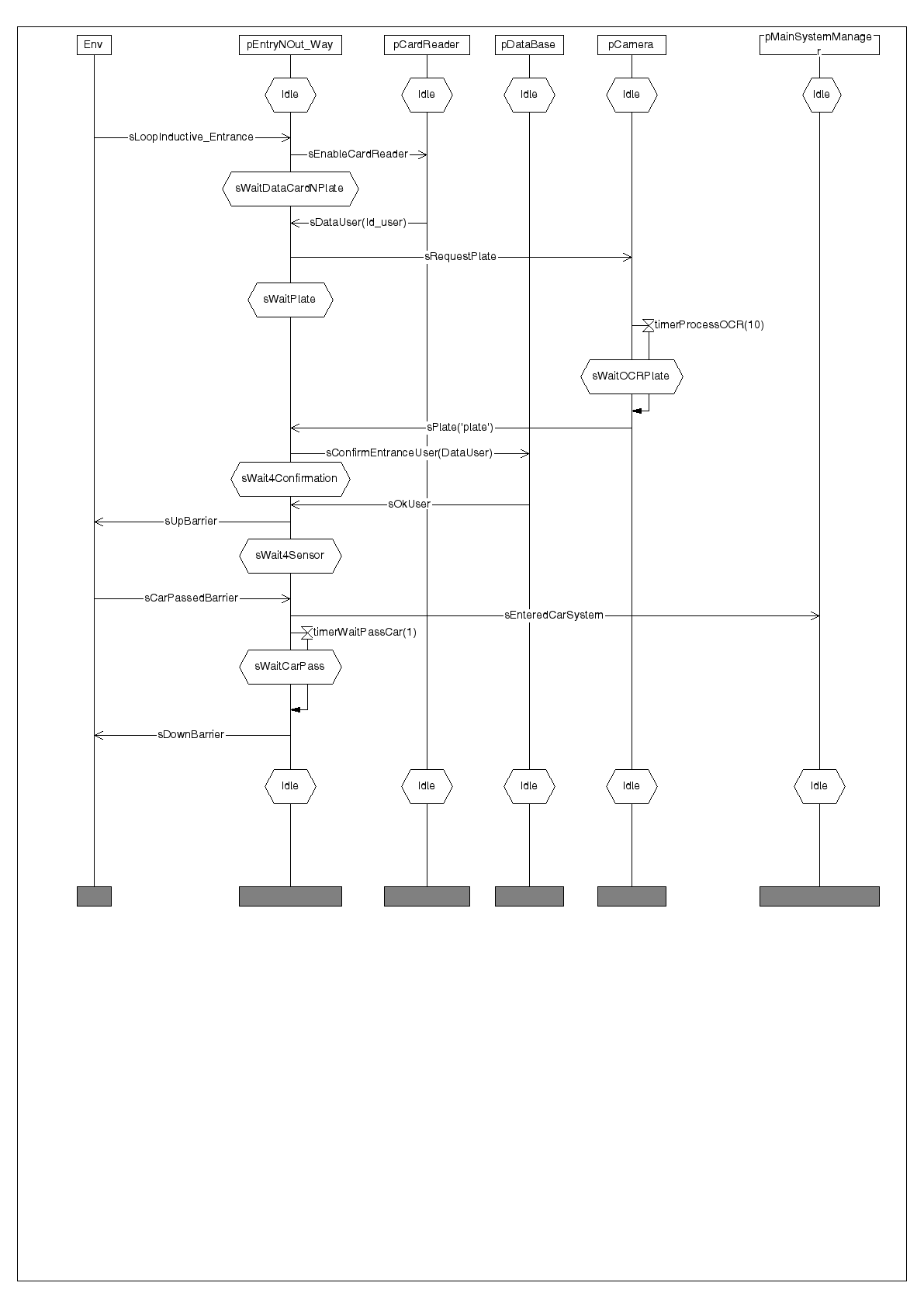


Figura 13. MSC ingreso de un vehículo al sistema de parqueo.

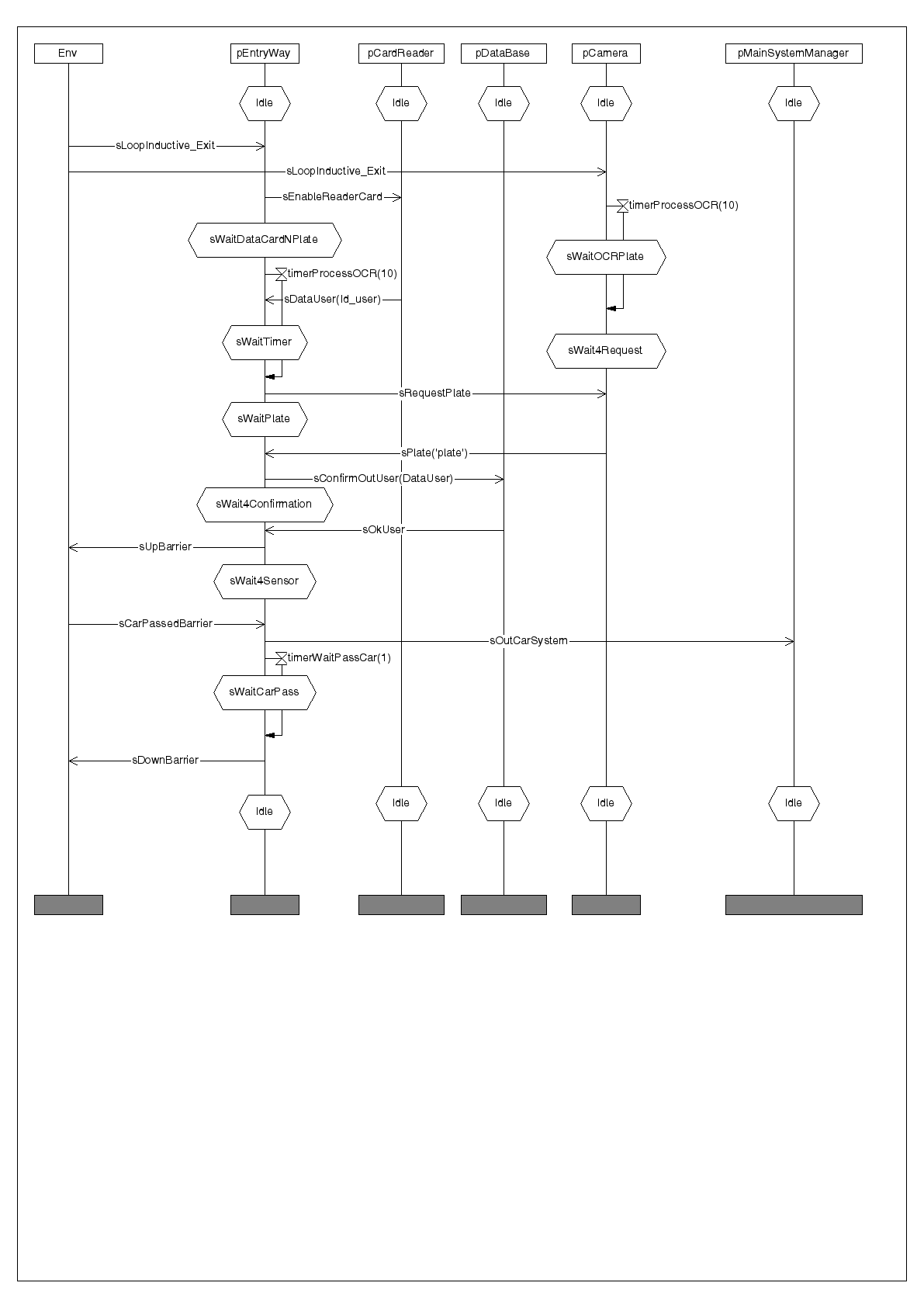


Figura 14. MSC Salida de un vehículo del sistema de parqueo.

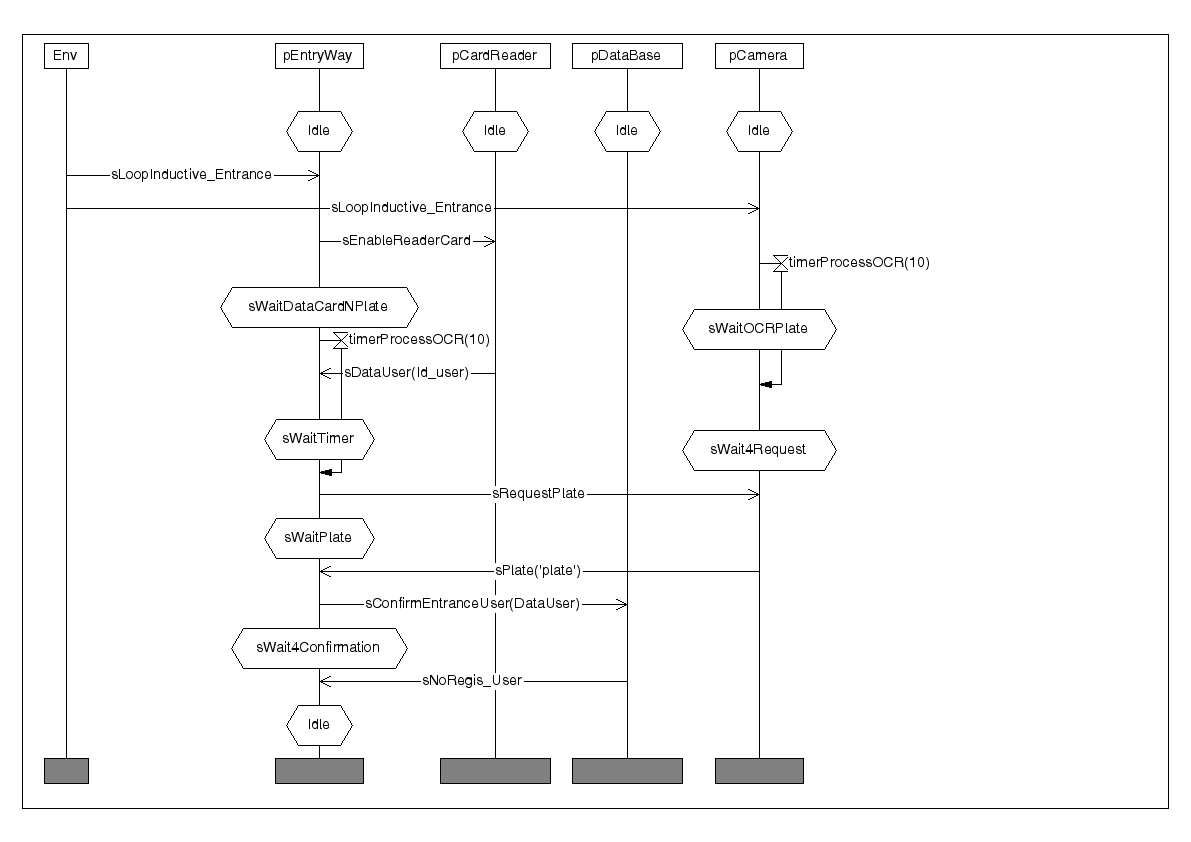


Figura 15. MSC usuario no autorizado intentando ingresar al sistema de parqueo de la PUJC.

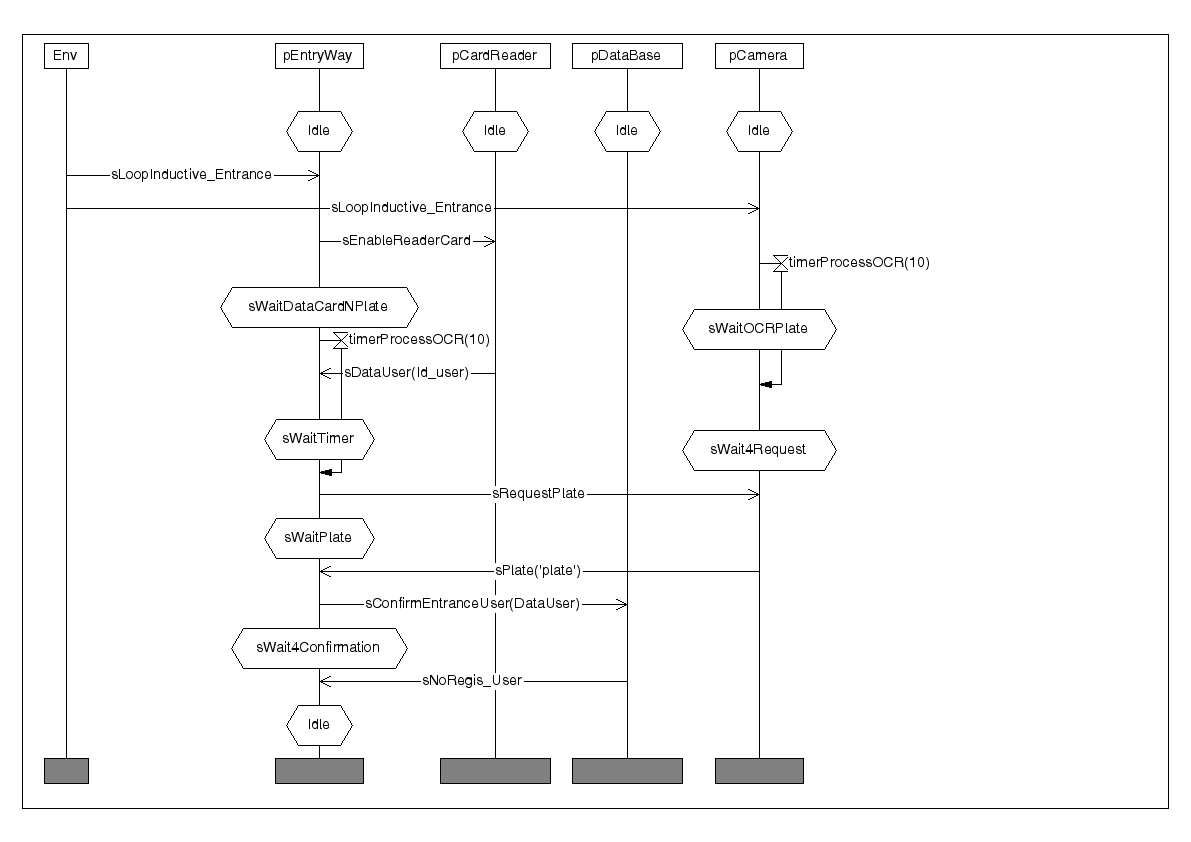


Figura 16. MSC usuario no autorizado intentando salir del sistema de parqueo de la PUJC.

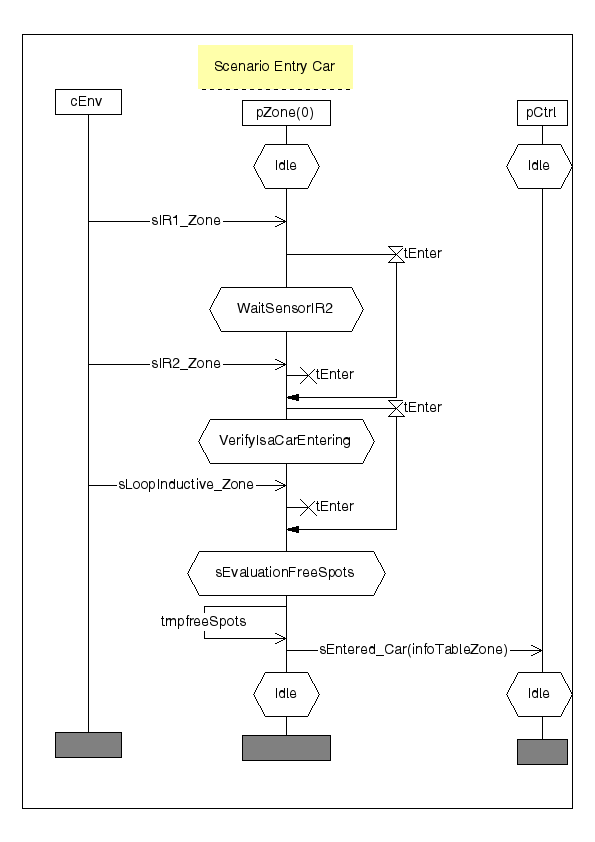


Figura 17. MSC usuario ingresando a una zona del sistema de parqueo.

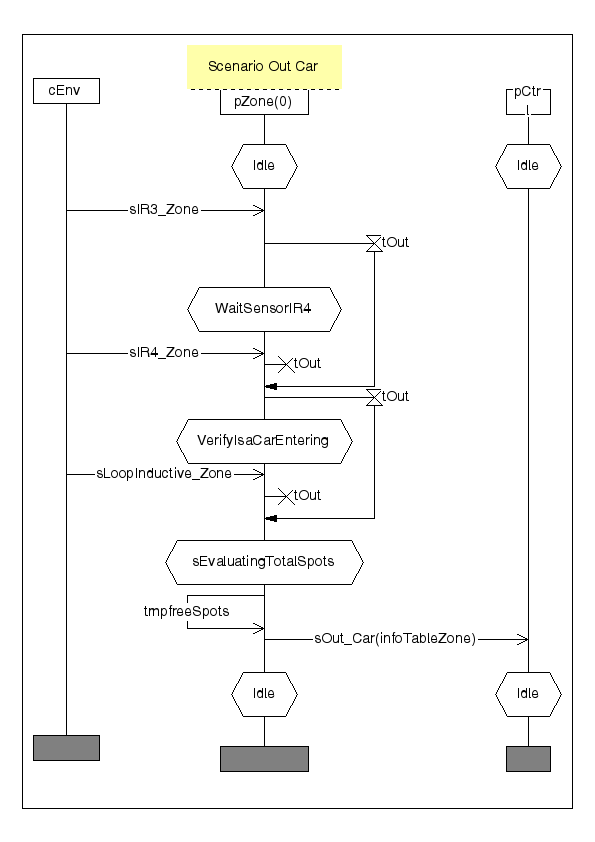


Figura 18. MSC usuario saliendo de una zona del sistema de parqueo.

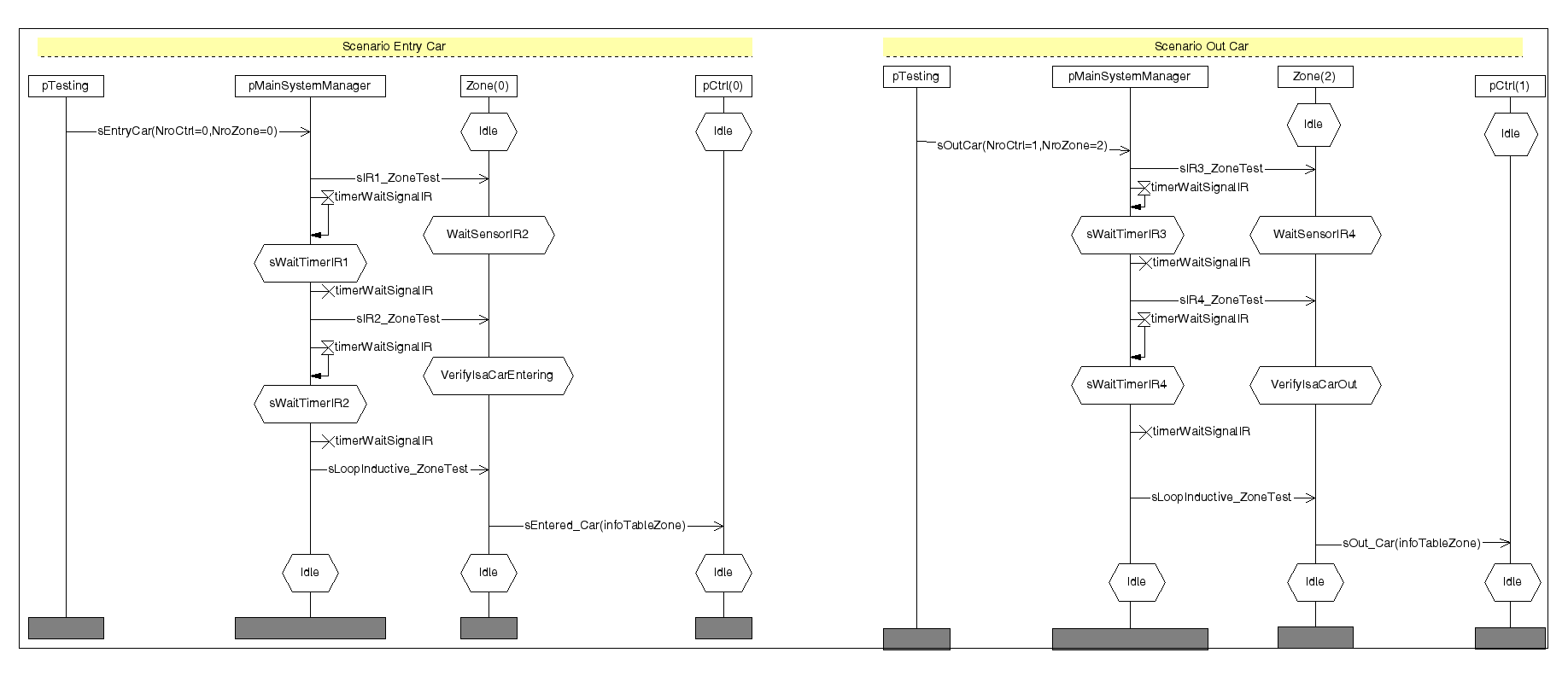


Figura 19. MSC Ingreso y salida de un vehículo a través del proceso Testing´

### Diseño usando SDL

Bottom-Up fue la estrategia de diseño usada para el desarrollo del sistema. Se inicia implementando el bloque BZone representado en la figura 20, dado que contiene procesos que poseen un bajo nivel de abstracción. Inicialmente se diseñó el proceso pZone el cual representa una zona de parqueo.

Como se había planteado en las especificaciones, el administrador puede crear una zona e inicializar sus parámetros los cuales son: plazas totales y libres. Ambos parámetros son subtipos del tipo de dato Integer, ver Anexo A para los tipos de datos usados en el modelado del sistema de parqueo. La zona tiene la facultad de reportar a su respectivo controlador el ingreso o salida de vehículos, y de enviar su información como una estructura de datos InfoZone, ver Anexo A, cada que haya un requerimiento por parte del controlador.

Las figuras 21, 22 y 23 representan la máquina de estados que se diseñó para detectar si un vehículo está por ingresar o salir de una zona del sistema de parqueo. Como se aprecia en la figura 22, cuando la zona no tenga plazas libres, freeSpots, el proceso pZone reportará a su respectivo proceso pCtrl que tiene cero plazas libres, así ingresen más vehículos a esta zona de esta forma se evitará enviar valores negativos de plazas libres. Lo anterior es similar a la figura 23 cuando la zona tenga todas sus plazas libres e intente salir un vehículo de ésta, se reportará que las plazas libres son iguales a las máximas permitidas o ajustadas por el administrador evitando enviar valores mayores a los permitidos por él sistema.

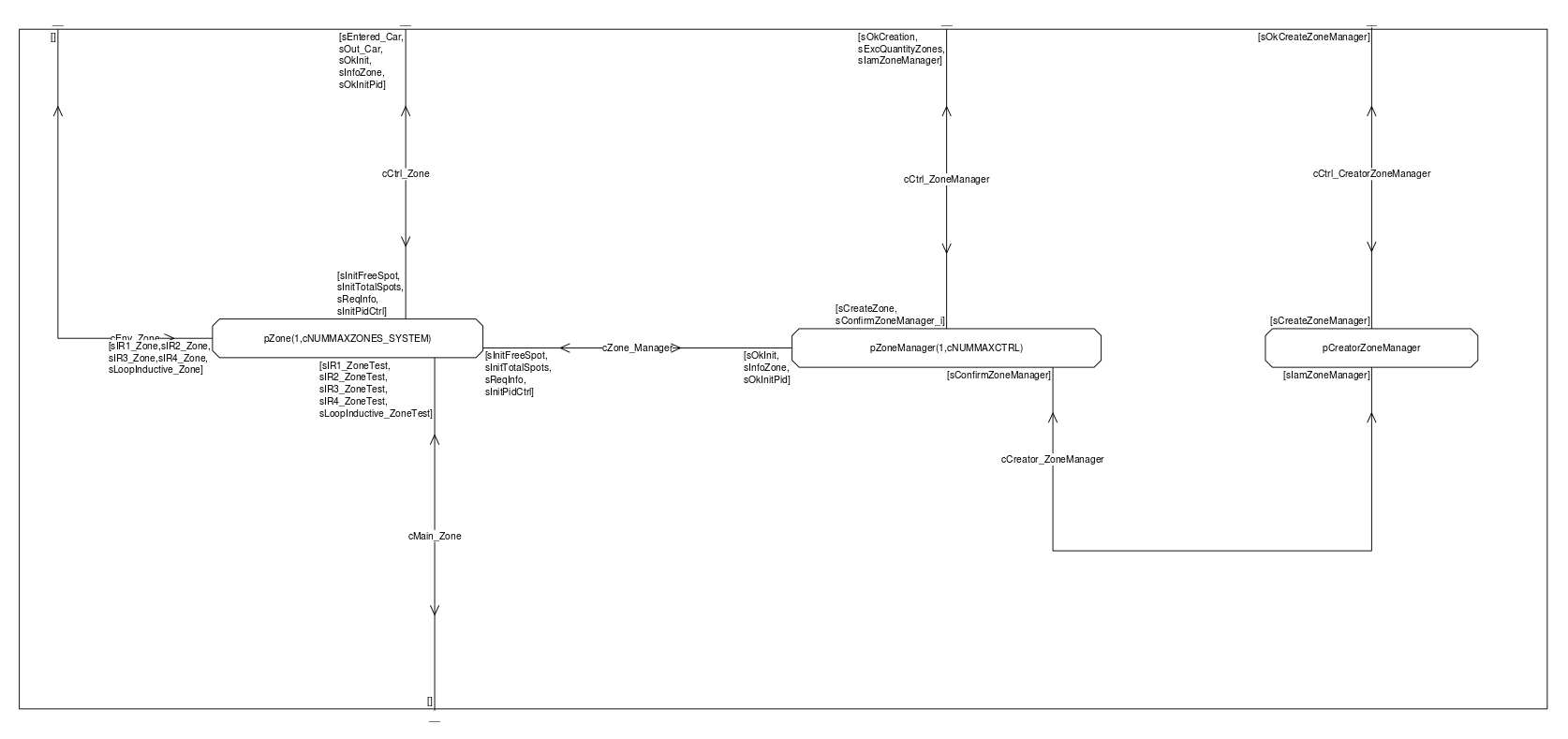


Figura 20. Arquitectura del bloque BZone

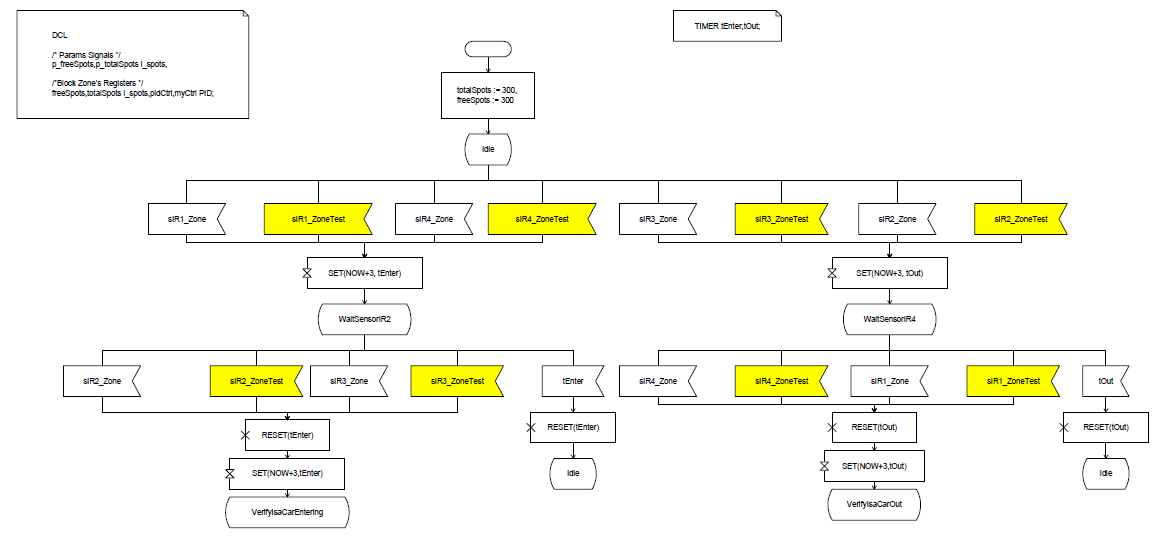


Figura 21. Máquina de estados del proceso pZone para el acceso o salida de un vehículo en una zona.

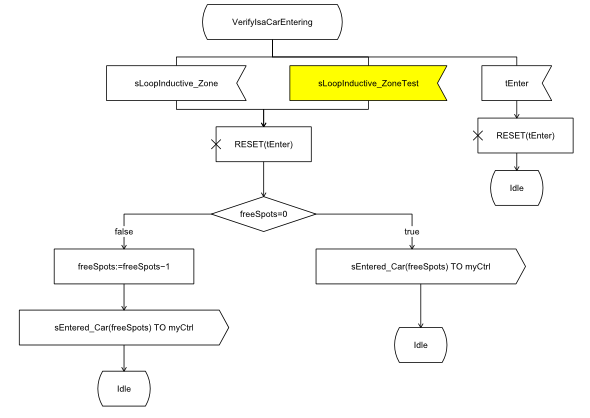


Figura 22. Continuación máquina de estados para el ingreso de un vehículo en una zona de parqueo.

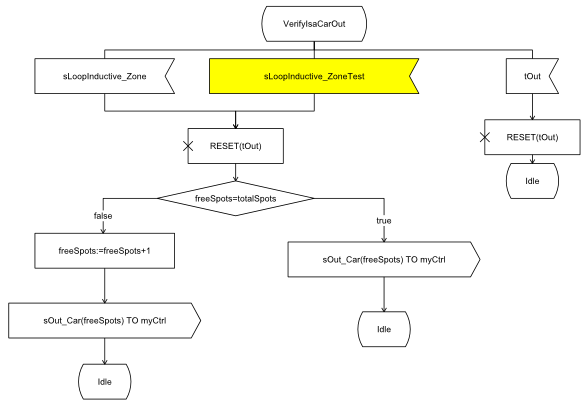


Figura 23. Continuación máquina de estados para la salida de un vehículo de una zona de parqueo.

Las señales de entrada resaltadas que se encuentran en la figura 21, son provenientes del proceso pMainSystemManager, nuevamente éste tipo de señales son usadas para implementar pruebas funcionales de caja negra, su utilidad se explicará en la siguiente sección.

La figura 21 muestra la implementación de dos temporizadores que tienen como función regresar al proceso pZone a un estado válido y que éste no se quede en uno de bloqueo. Un ejemplo en el cual éstos temporizadores serían útiles es cuando el proceso pZone reciba la señal del sensor sIR1\_Zone y posteriormente la señal sIR2\_Zone, lo anterior se consideraría que un vehículo está por ingresar a dicha zona, pero si el causante de la interrupción fue una persona nunca llegaría la señal sLoopInductive\_Zone, por lo cual el proceso pZone quedaría en el estado VerifyIsaCarEntering, si ésta señal nunca llega después de cierto tiempo, el temporizador coloca al proceso nuevamente en el estado Idle dónde puede efectuar otras funciones sin bloqueo alguno.

La figura 24 se muestran los estados y las transiciones que el proceso pZone efectúa para la inicialización de plazas totales, plazas libres, requerimientos de información y la asignación del identificador de su controlador.

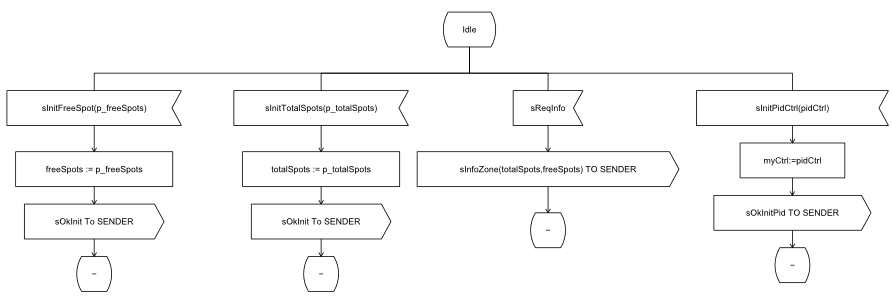


Figura 24. Máquina de estados para la inicialización de parámetros de una zona y solicitud de requerimiento de información a ésta.

Una vez implementada la máquina de estados del proceso pZone, se implementa el proceso pZoneManager que está encargado de instanciar más procesos pZone y asignarle a éste su respectivo controlador de zonas.

En las figuras 25 y 26 se pueden apreciar la máquina finita del proceso pZoneManager, inicialmente el sistema permanece en el estado Idle y está habilitado para recibir las siguientes señales: sCreateZone, sConfirmZoneManager y sConfirmZoneManager\_i. La primera señal tiene asociado dos parámetros de tipo i\_spots que son subtipos del tipo de dato Integer, básicamente cuando el proceso pZoneManager recibe esta señal crea un instancia de una zona, si el sistema no tiene más capacidad para zonas el agente pZoneManager enviará una señal sExcQuantityZones a su respectivo controlador indicando que no es posible anexar más zonas, de lo contrario creará un proceso pZone y efectuará el intercambio de señales con éste para inicializar sus parámetros.

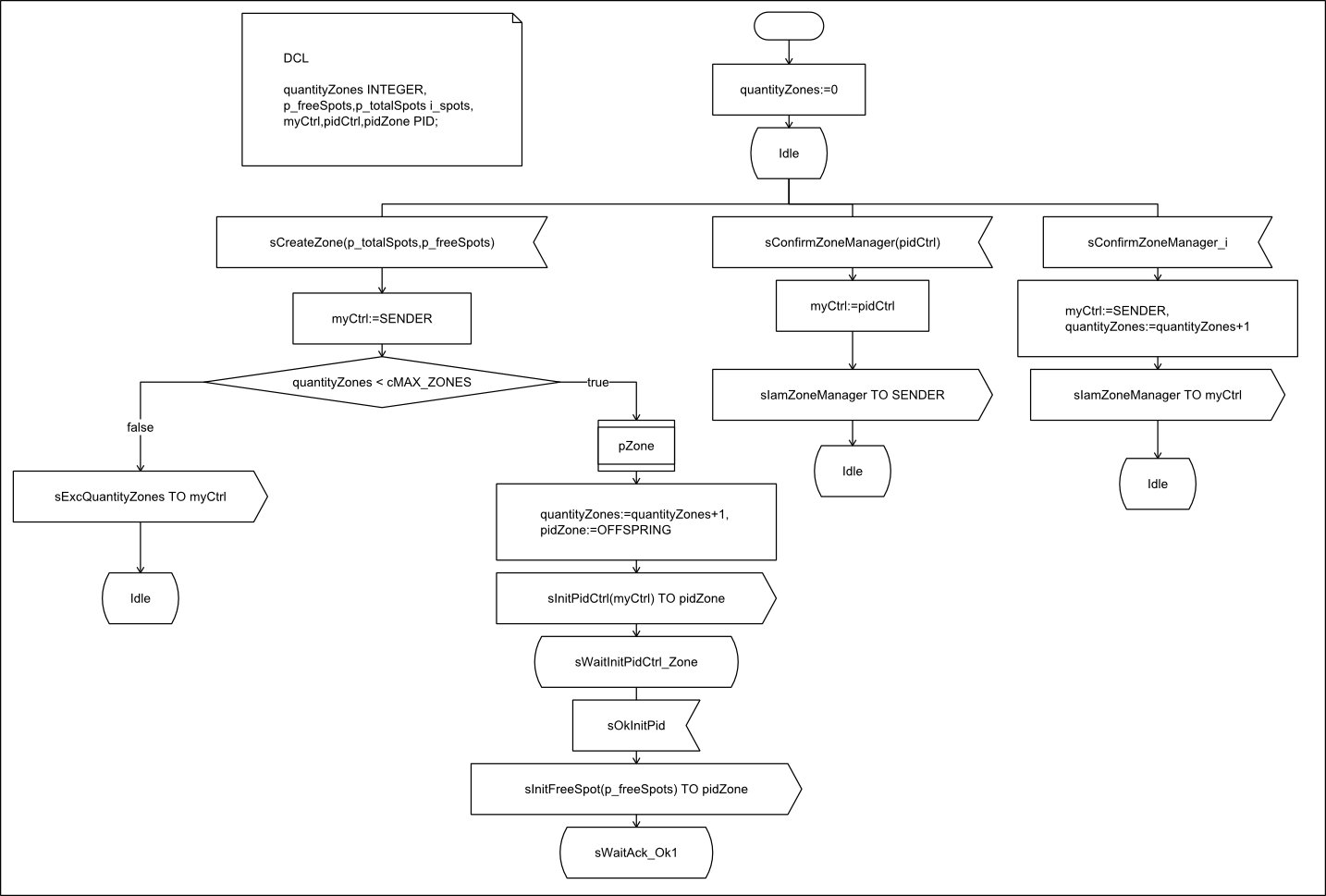


Figura 25. Máquina de estados del proceso pZoneManager para la creación de zonas e inicialización de su respectivo controlador de zonas.

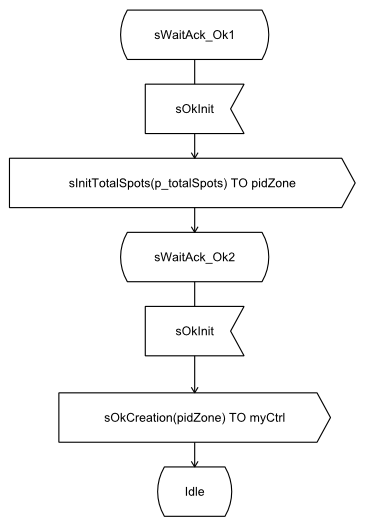


Figura 26. Continuación máquina de estados para la creación de una zona de parqueo.

La figura 27 representa la máquina de estados finita del proceso pCreatorZoneManager, la única función que tiene este proceso es de instanciar procesos pZoneManager y asignarle a éste su respectivo controlador de zonas.

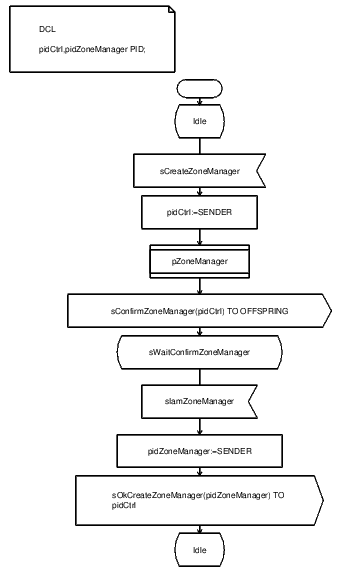


Figura 27. Máquina de estados finita del proceso pCreatorZoneManager.

La figura 28 muestra la arquitectura del bloque UnmodeledProcesses. En esta arquitectura se puede apreciar que los procesos pCardReader que representa el lector de carnés y pCamera que representa la cámara, solicitan al ambiente o entorno los datos que deberían de entregar al proceso pEntryNOut\_Way, de esa forma éstos procesos solo sirven para mostrar los canales y las conexiones a otros procesos que deberían de ser usados en el momento de ser modelados en detalle.

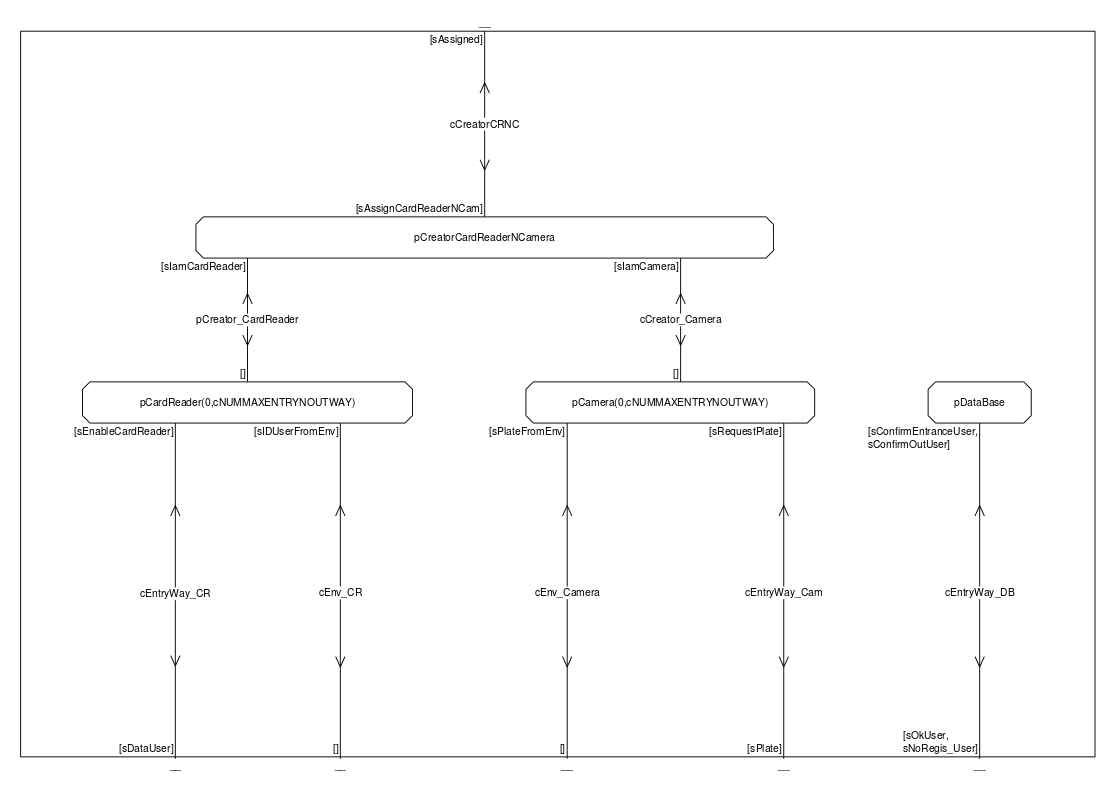


Figura 28. Arquitectura del bloque Unmodeled Processes

Para el modelado del repositorio de información se ha construido un arreglo de datos que contiene un identificador que tiene asociada una única placa, actualmente el sistema cuenta con 40 usuarios registrados, pero no se limita a esta cantidad en su implementación ni tampoco hace que se desvíe el objetivo de este trabajo de grado. La figura 29 representa la máquina de estados del proceso pDataBase que representa los repositorios de información del sistema.

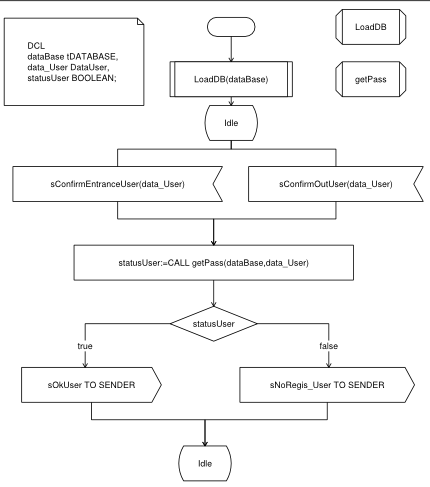


Figura 29. Máquina de estados del proceso pDataBase

La figura 30 representa el procedimiento usado para cargar los datos al repositorio de datos, y la figura 31 representa el procedimiento que valida si un usuario tiene acceso al sistema de parqueo, si su identificador obtenido del lector de carnés y la placa tomada por la cámara se encuentran en dicho repositorio. Este tipo de búsquedas se podrían ampliar estableciendo algunas políticas de acceso, por ejemplo:

* Que un usuario con su identificador pueda tener asociado varias placas de vehículos.
* Que el usuario tenga acceso al sistema si y solo si sus vehículos aparece en el repositorio que no están dentro del sistema de parqueo.
* Que un mismo usuario no pueda tener acceso al parqueo con otro vehículo registrado si ya ha ingresado otro.

Las opciones descritas anteriormente serían parte de las especificaciones para el modelado y diseño de los repositorios del sistema de parqueo.

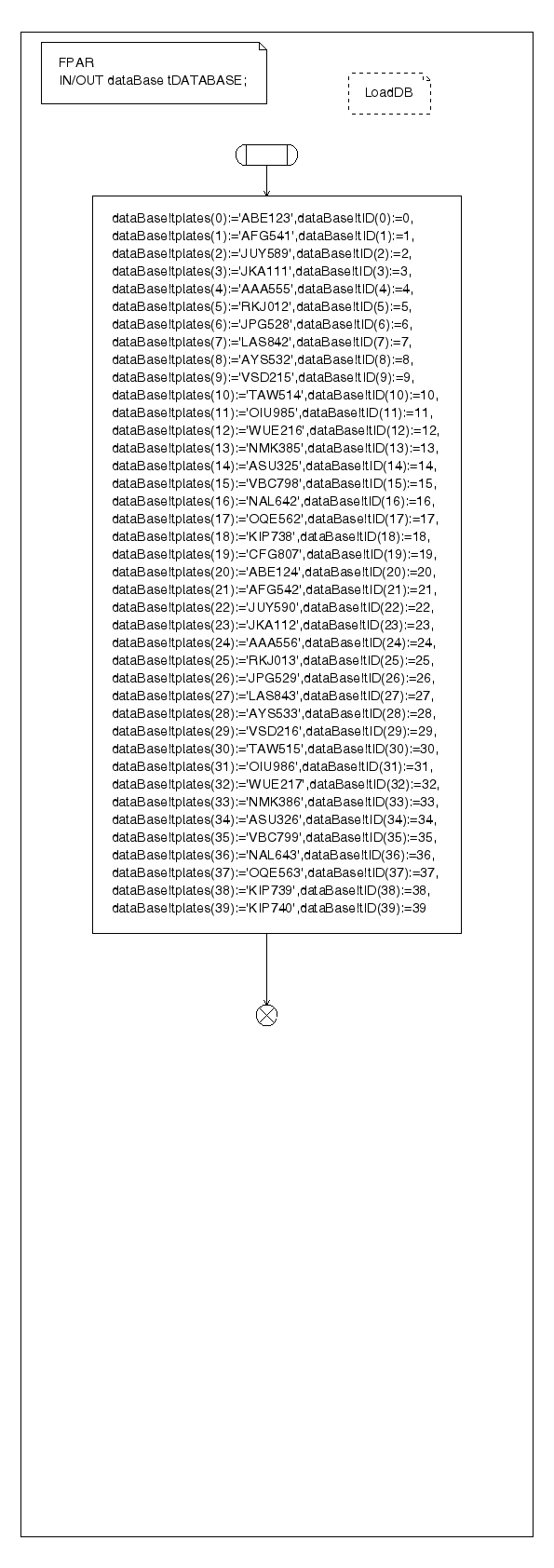


Figura 30. Procedimiento para cargar los datos al repositorio del sistema de parqueo.

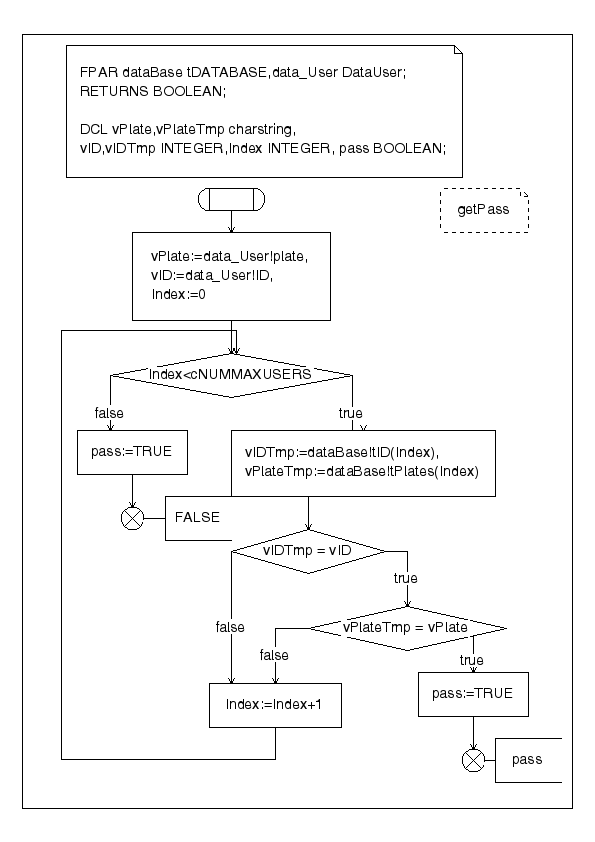


Figura 31. Procedimiento para validar si un usuario tiene acceso o salida del sistema de parqueo.

Para más información del diseño de las otras máquinas de estado véase el Anexo C.

## Pruebas

Al igual que las estrategias de diseño mencionados en la sección del modelado, éstas pueden ser empleadas en la fase de pruebas de un modelo, en este caso se va a hacer uso de la estrategia Bottom-Up. La herramienta RTDS posee un módulo que permite hacer pruebas funcionales de caja negra, la forma que se puede evaluar si una prueba es satisfactoria puede ser comparando el MSC que se genera al estimular el sistema frente al MSC de la especificación del mismo escenario

La herramienta RTDS permite crear una interfaz gráfica como emisor y receptor de señales al sistema que se desea probar, su implementación se explicará más adelante.

Una de las ventajas que tiene el módulo de pruebas de la herramienta RTDS es que permite evaluar un modelo a través de un escenario específico de forma rápida y sencilla, pero carece de automatización en cuanto a los estímulos inyectados al sistema bajo prueba. Por otro lado, el lenguaje TTCN-3 ha sido diseñado para hacer pruebas funcionales de caja negra y tiene la ventaja de poder automatizarlas generando una mayor cantidad de estímulos, lo que se traduce a obtener sistemas más seguros.

La figura 32 muestra el simulador de modelos descritos en SDL que tiene la herramienta RTDS.

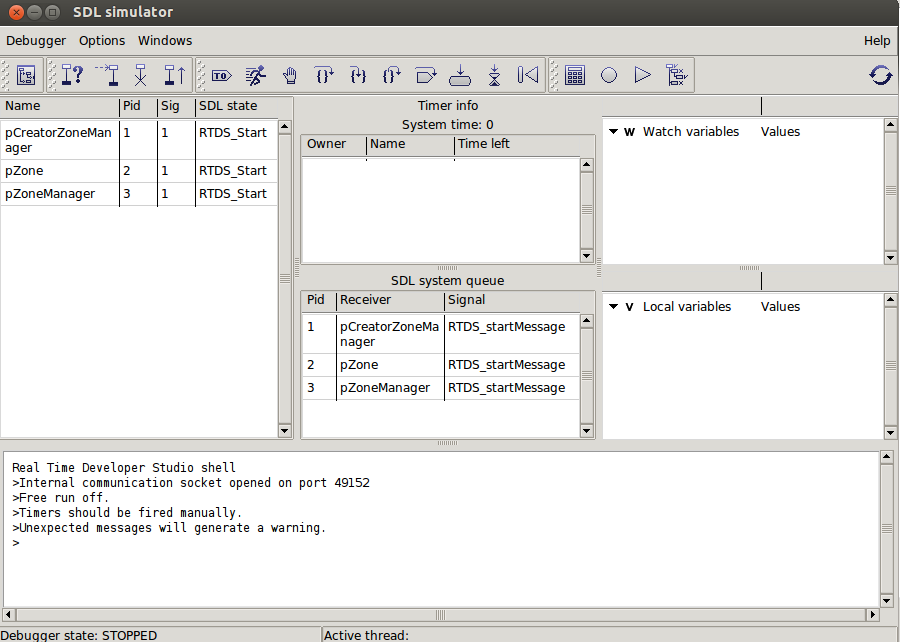


Figura 32. Interfaz gráfica para el módulo de pruebas en la herramienta RTDS

En el botón **TO** se pueden seleccionar las señales que se desean enviar a procesos específicos, como se muestra en la figura 33.

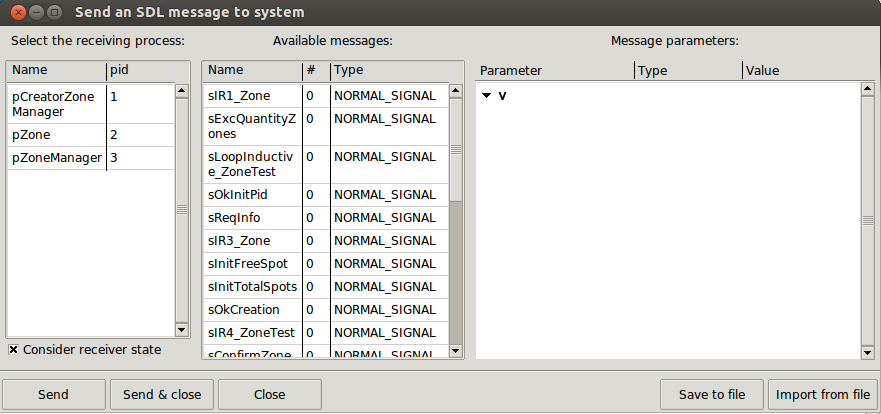


Figura 33. Interfaz envío de señales a procesos a través de la herramienta RTDS.

Como se describió en la parte del modelado del sistema de parqueo, el bloque BZone tiene tres procesos entre los cuales se encuentra el proceso pCreatorZoneManager que está encargado de crear procesos de tipo pZoneManager. La figura 34 muestra el resultado de un escenario de prueba de creación de un proceso pZoneManager, la forma de verificar que el diseño de las máquinas de estados quedó conforme a los requerimientos es comparando el MSC resultante de la prueba con el MSC de la especificación.

Lo que muestra la figura 34 es básicamente que el proceso pCreatorZoneManager inicialmente se encuentra en el estado Idle y está esperando por la señal sCreateZoneManager, una vez reciba esta señal crea el proceso pZoneManager asignándole su proceso controlador y pasa al estado sWaitConfirmZoneManager.

Como el entorno solicitó la creación de un pZoneManager entonces su PID por defecto es cero, por eso el parámetro que corresponde al identificador del controlador de zonas tiene el mismo valor. Una vez el proceso pZoneManager haya asignado el identificador de su controlador, enviará una señal sIamZoneManager a su proceso creador el cual se encuentra en un estado permitido para recibir dicha señal y retorna al proceso que solicitó inicialmente la creación la señal sOkCreateZoneManager enviando como parámetros de esa señal el identificador del nuevo proceso creado. Ambos procesos terminan su interacción y quedan en el estado Idle.

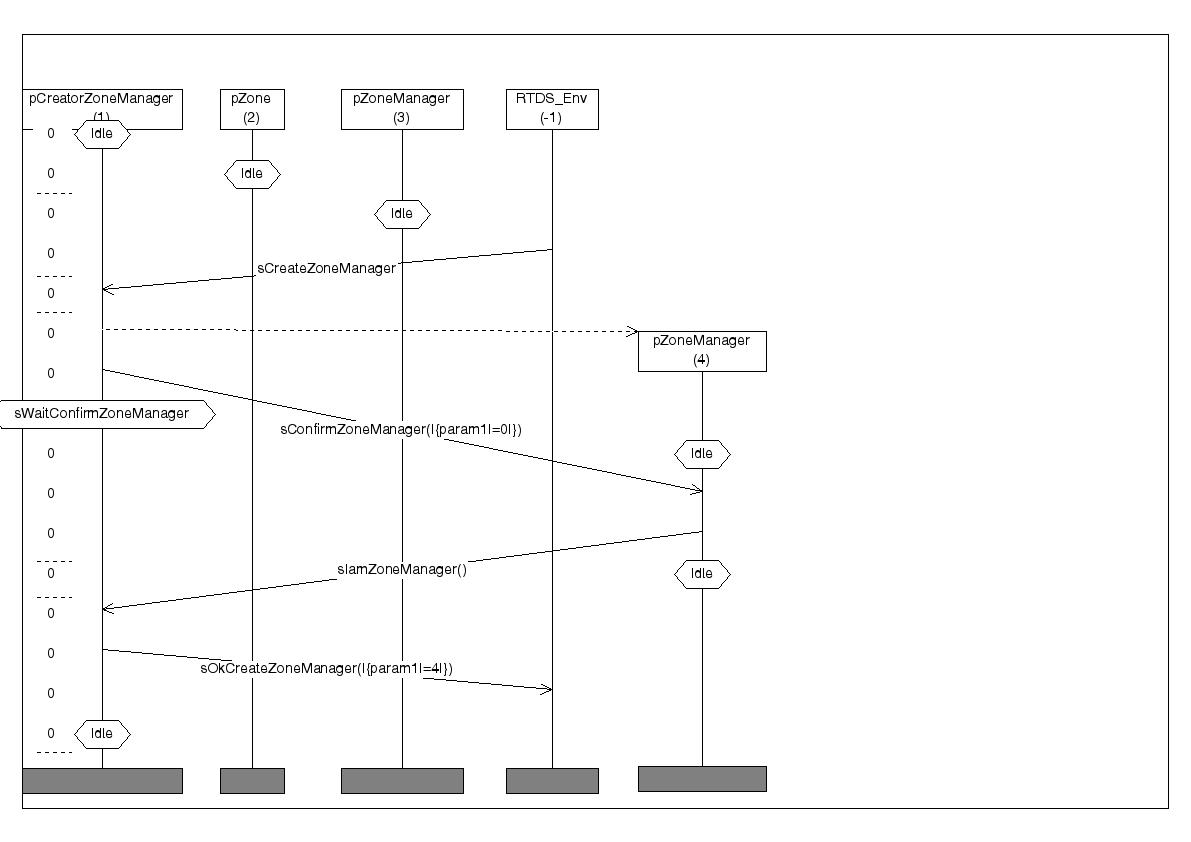


Figura 34. Resultado de la prueba de crear una instancia de un proceso pZoneManager

La tabla 5 muestra los resultados de las pruebas realizadas al bloque BZone del sistema de parqueo usando la herramienta RTDS. Por defecto la condición inicial del bloque BZone, siempre contará con una instancia del proceso pZone, pZoneManager y pCreatorZoneManager. Adicionalmente la zona de parqueo contará con 300 plazas libres y 300 totales.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Número de prueba | Propósito | Pre-condición | Post-Condición | Proceso(s) donde se efectúa la prueba |
| 1. Figura 35 | Probar que el proceso pZoneManager puede crear satisfactoriamente un proceso pZone. | Por defecto. | El sistema cuenta con dos instancias del proceso pZone y una instancia tanto del proceso pZoneManager como del proceso pCreatorZoneManager. | pZoneManager |
| 1. Figura 36 | Probar que el proceso pZoneManager puede instanciar un número máximo de zonas y reportar cuando se vaya a sobrepasar dicho límite. | Por defecto. | El sistema cuenta con el número máximo de zonas que puede crear el proceso pZoneManager, en este caso 3, al tratar de crear otra zona se generará una señal que indica que no es posible crear más procesos pZone. | pZoneManager |
| 1. Figura 37 | Probar que se puede inicializar las plazas libres y totales de una zona del sistema de parqueo. | Por defecto. | El sistema cuenta con dos zonas, a la nueva zona se inicializa sus zonas totales y libres. | pZone y pZoneManager |
| 1. Figura 38 | Probar que la zona de parqueo es capaz de reportar una cifra correcta de plazas libres después de haber entrado un vehículo a ésta. | Por defecto. | La zona reportará que han ingresado dos vehículos por lo tanto su valor de plazas libres es de 298. | pZone |
| 1. Figura 39 | Probar que la zona reporta un número correcto de plazas libres después de ingresar y salir vehículos de dicha zona. | Por defecto. | La zona reportará que han ingresado dos vehículos y salido uno. Por lo tanto, su valor de plazas libres es de 299. | pZone |

Tabla 5. Resultado de las pruebas realizadas en el bloque BZone

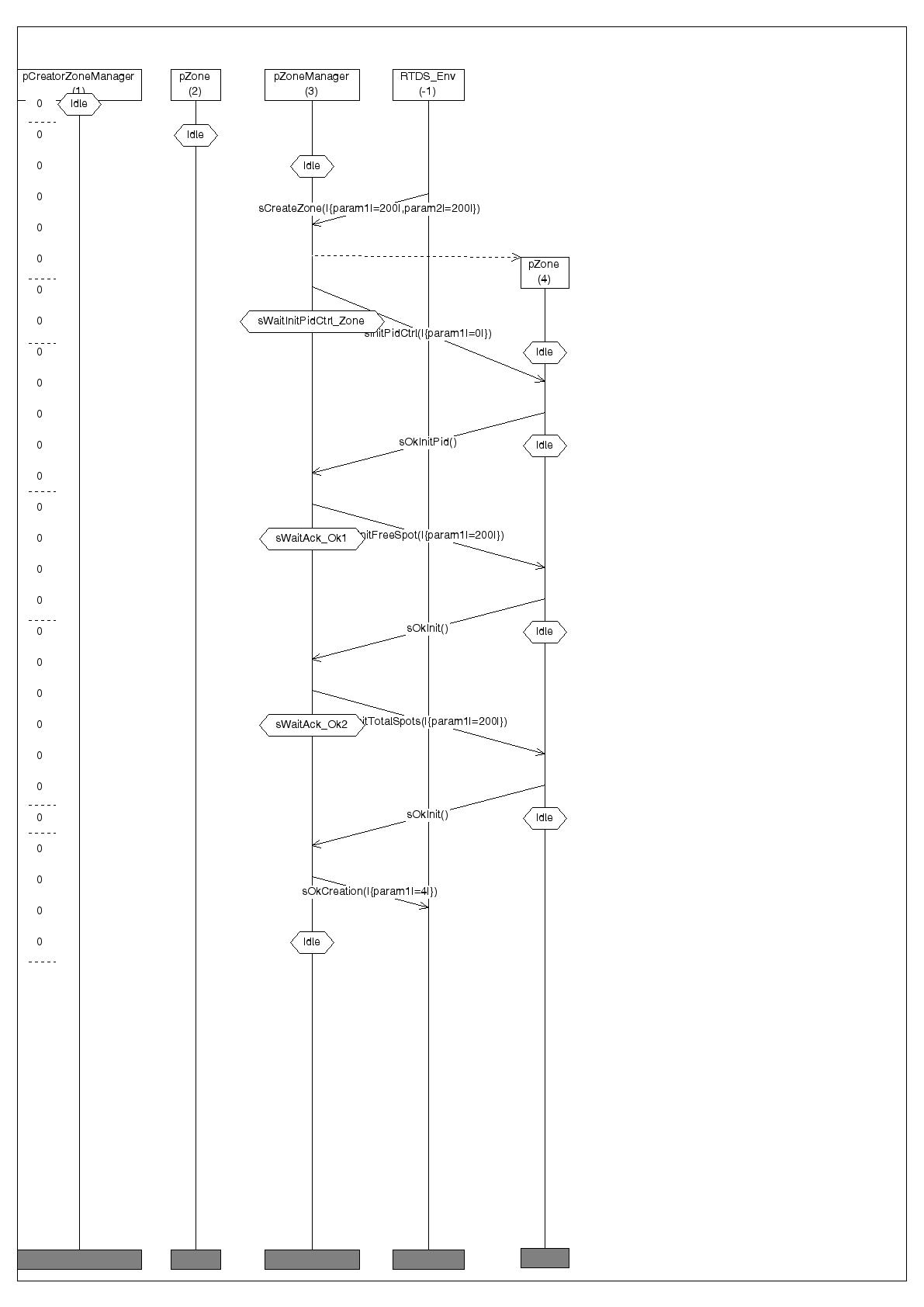


Figura 35. MSC resultado de la prueba de creación de una zona de parqueo.

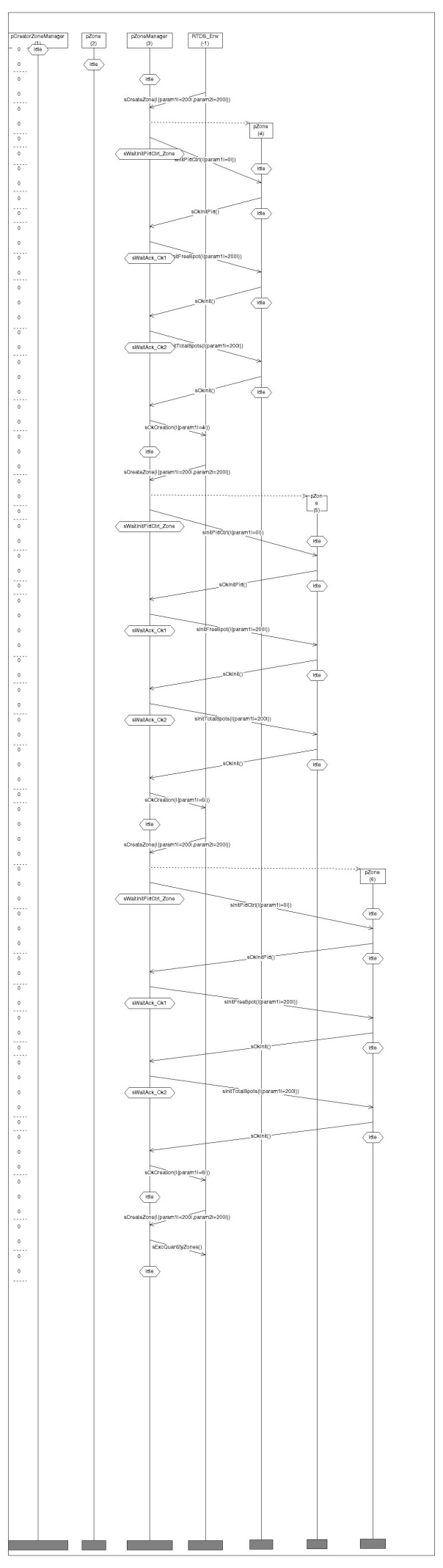


Figura 36. MSC resultado de la prueba de creación de número máximo de pruebas permitido por un proceso pZoneManager.

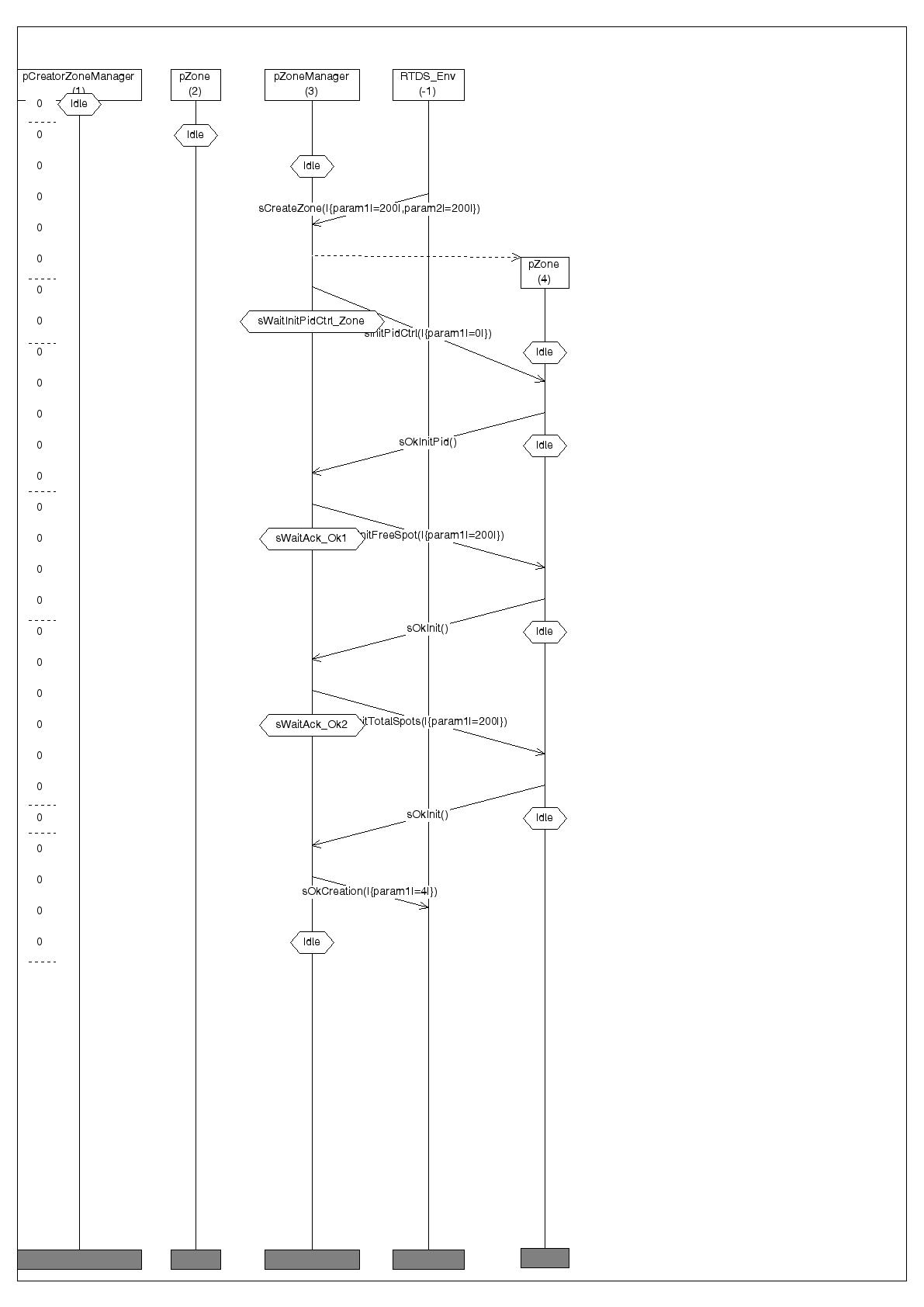


Figura 37. MSC resultado de la prueba de inicializar las plazas totales y libres en una zona de parqueo.

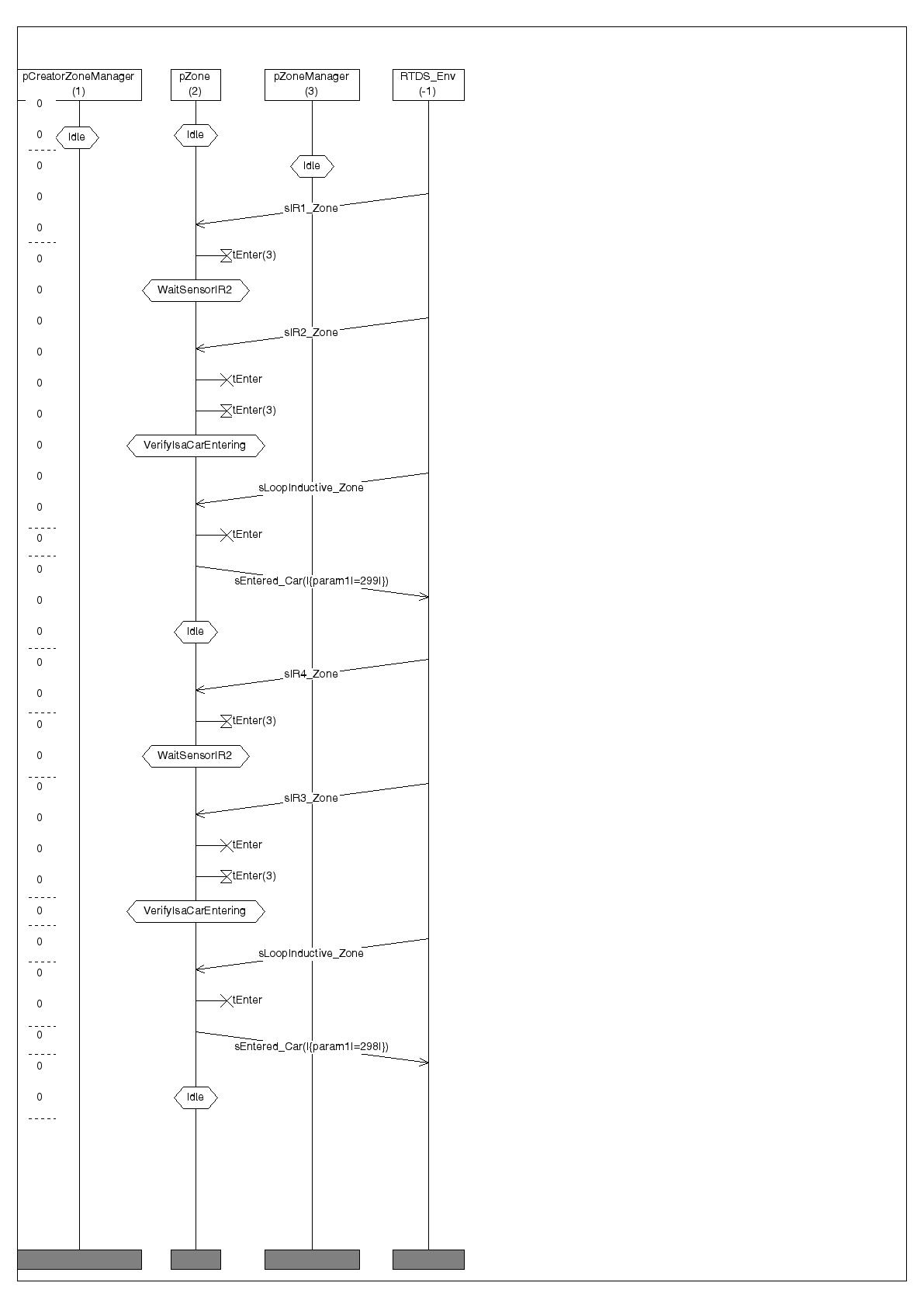


Figura 38. MSC resultado de la prueba del ingreso de un vehículo en una zona de parqueo.

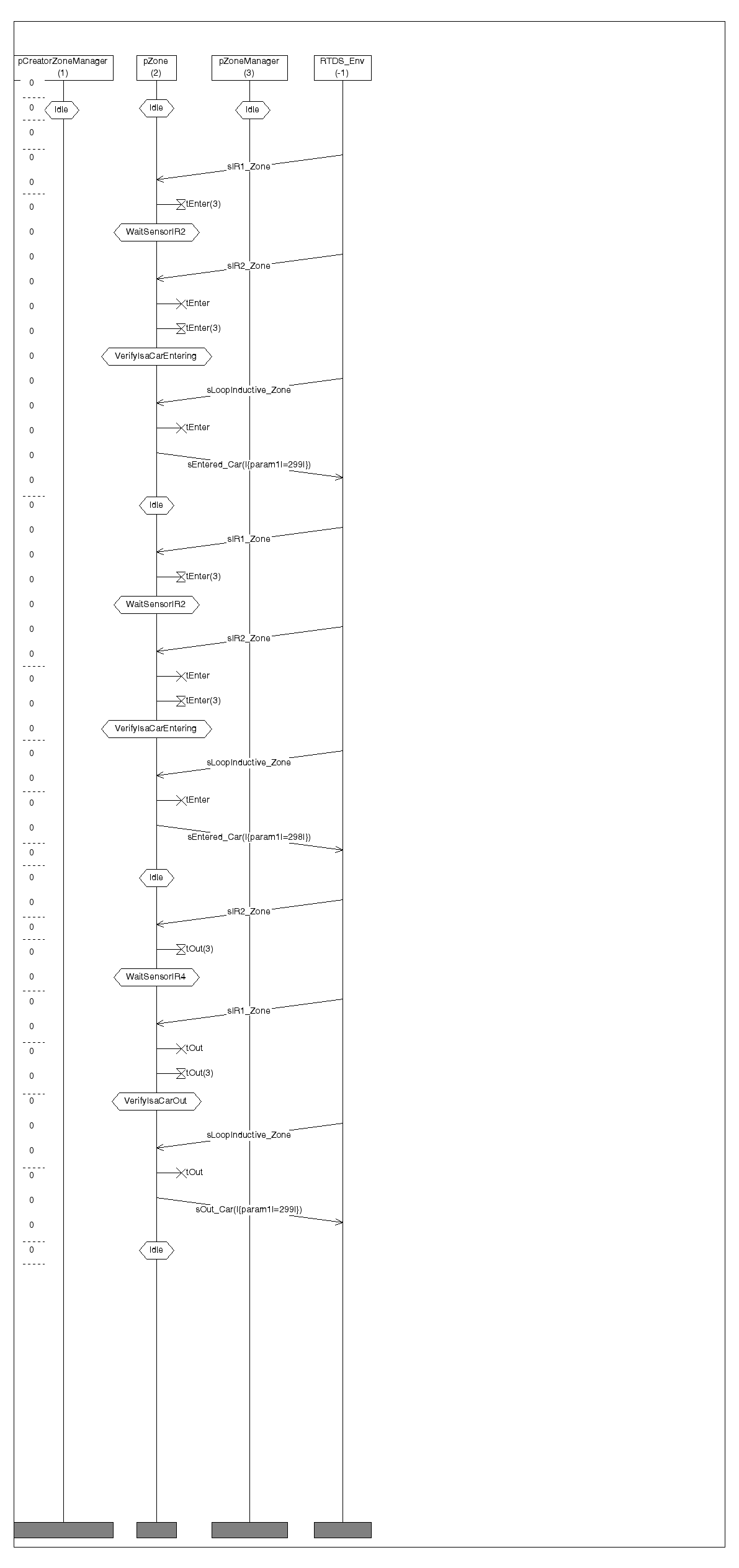


Figura 39. MSC resultado de la prueba de la salida de un vehículo de la zona de parqueo.

Aprovechando la característica que tiene la herramienta RTDS de crear una interfaz gráfica como fuente de estímulos al sistema bajo pruebas, se diseña una como se muestra en la figura 40.

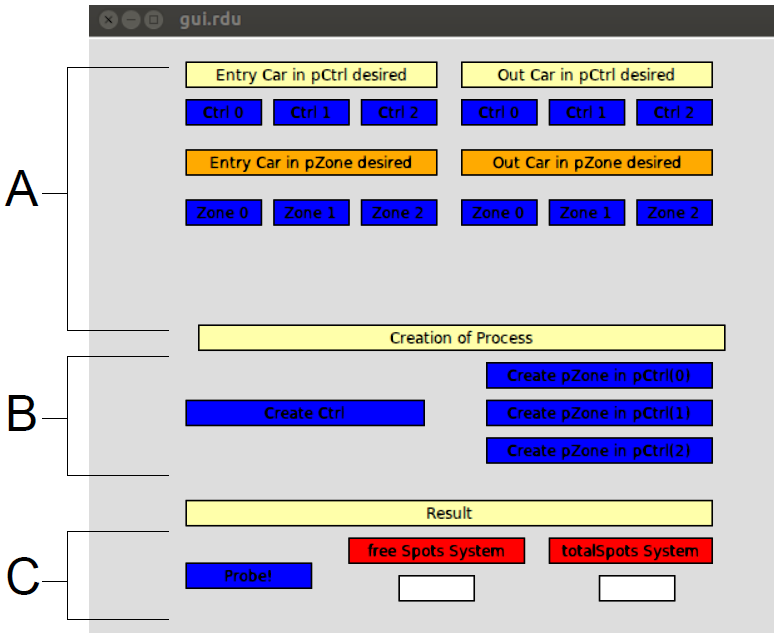


Figura 40. Diseño interfaz gráfica para la realización de pruebas.

Como se había comentado anteriormente en el diseño del sistema de parqueo de la PUJC se ha modelado un proceso llamado pTesting, como acople para hacer pruebas al modelo a través de una interfaz gráfica. La limitación que tiene la interfaz gráfica es que no permite ajustar todos los parámetros de una señal, por lo cual si se deseará simular el ingreso de un vehículo a la zona 0 del controlador 0 no sería posible hacer esto con único objeto gráfico de tipo botón; por ende, es indispensable que el proceso pTesting capture los valores del número de controlador y el número de zona dónde se desea simular un ingreso de un vehículo en dos diferentes estados. Similarmente es el proceso de captación de datos para la simulación de la salida de un vehículo de una zona de parqueo.

Una vez el proceso pTesting tenga el número de controlador y la zona a hacer simulada, enviará la señal correspondiente al sistema para que simule en la zona y controlador específico la entrada o salida de vehículos.

Esta interfaz se ha diseñado para probar que el sistema modelado puede crear instancias de controladores de zonas y anexar a éstos nuevas zonas. Adicionalmente cuenta con la posibilidad de ingresar y sacar automóviles de una zona y un controlador específico.

La parte A de la interfaz tiene la funcionalidad de ingresar un automóvil a una zona de un controlador específico. La manera de hacerlo es seleccionando un controlador ubicado debajo de la etiqueta ***Entry Car in pCtrl desired***. Una vez determinado el controlador por el cual un vehículo ingresó, se selecciona la zona que reportó el ingreso del vehículo ubicado bajo la etiqueta ***Entry Car in pZone desired***. De forma análoga el proceso de sacar un vehículo de una zona de parqueo es el mismo únicamente se selecciona el controlador que está bajo la etiqueta ***Out Car in pCtrl desired*** y la zona a que está abajo de la etiqueta ***Out Car in pZone desired.***

La parte B de la interfaz corresponde a la funcionalidad de crear nuevos controladores y zonas. La forma de crear un controlador es seleccionando el botón ***Create Ctrl***, en ese momento se enviará una señal al sistema con el propósito de crear un nuevo proceso pCtrl. Para anexar una nueva zona a un controlador específico, se selecciona el botón ***Create pZone in pCtrl(n)*** donde n corresponde al controlador, para fines prácticos el sistema cuenta con 3 controladores de zonas y cada uno puede tener a su mando hasta 3 zonas.

La parte C de la interfaz corresponde a la suma total de las plazas libres y totales que tiene cada controlado para verificar que el sistema satisface las pruebas. Bajo la etiqueta ***free Spots System*** se detallan las plazas libres totales de todos los controladores y bajo la etiqueta ***total Spots System*** se muestra la suma de las plazas totales de todos los controladores. La información anterior es indicada por medio de la interfaz gráfica si y solo si es presionado el botón ***probe!*** después de haber corrido el sistema en el simulador de RTDS.

Indudablemente los entornos gráficos como medio de acople para realizar pruebas a los sistemas muchas veces ayudan al desarrollador a obtener información del diseño de una forma más amigable. La herramienta RTDS ofrece éste tipo de módulos pero carece de más objetos gráficos para ampliar el nivel de la prueba, lo anterior implicó establecer valores constantes a algunos parámetros de las señales a ser enviadas al sistema bajo prueba a través de la interfaz gráfica. Por ejemplo: Al momento de crear una zona en un controlador el administrador establece el controlador de zonas el cual va a ser ampliado con una nueva zona, la cantidad de plazas totales y finalmente se define la cantidad de plazas libres. Dado que no existe un objeto gráfico que permita inicializar una señal con más de dos parámetros, se definió que: todas las zonas del controlador 0 tendrán 200 plazas libres y totales, todas las zonas del controlador 1 tendrán 100 plazas libres y totales y finalmente todas las zonas del controlador 2 tendrán 50 plazas libres y totales.

La figura 41 detalla el número de plazas totales de los controladores del sistema cuando se ha sometido al siguiente escenario:

* El sistema cuenta con tres controladores y 9 zonas en total.
* Inicialmente no han ingresado ni salido vehículos.
* La cantidad de plazas libres y totales son las mismas, y corresponden a la sumatoria de las plazas libres de todos los controladores, El controlador 1 cuenta tanto con 700 plazas libres como totales, el controlador 2 cuenta con 300 plazas tanto libres como totales y finalmente el controlador 3 cuenta con 150 plazas tanto libres como totales.
* Ingresan 4 carros a la zona 0 del controlador 0.
* Ingresa 1 carro tanto a la zona1 como a la zona 2 del controlador 0.
* Ingresa 2 carros a la zona 0, 3 carros a la zona 1 y 1 carro a la zona 2; todas las zonas pertenecen al controlador 1.

El resultado esperado de plazas libres y totales que se desea ver en la interfaz gráfica son: 1138 y 1150 respectivamente.

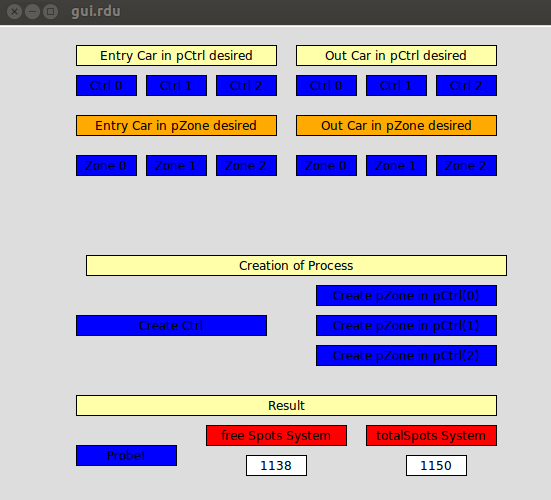


Figura 41. Resultado de las plazas totales y libres inyectando datos al sistema bajo prueba desde la interfaz gráfica.

Hasta el momento el diseño del sistema de parqueo ha sido evaluado por medio de pruebas funcionales de caja negra usando el simulador de la herramienta SDL. A pesar que las pruebas previas cumplen con el propósito de encontrar errores en el diseño del sistema, es indispensable inyectar el sistema bajo prueba estímulos más dinámicos y de forma automática. En este momento la potencia del lenguaje TTCN-3 para hacer pruebas funcionales de caja negra se puede aprovechar al máximo.

A partir de los sistemas de registro de vehículos que posee el sistema de parqueo de la PUJC se obtuvieron los datos de ingreso y salida de dicho sistema. Un ejemplo de la forma de los datos obtenidos se muestra en la figura 42.

Figura 42. Formato de Ingreso y salida de vehículos del sistema de parqueo de la PUJC

|  |  |
| --- | --- |
| Hora | Cantidad de vehículos |
| 06:35 | 30 |
| 06:40 | 26 |
| 06:45 | 36 |
| 06:55 | 37 |

Los procesos que serán evaluados pertenecen al bloque ParkingLotSystem. Debido que la estructura de información mostrada en la tabla 6 no muestra la zona ni su respectivo controlador en el cual un vehículo ha ingresado o salido de dicha zona, entonces se implementó una fórmula en el Software ***Office Calc*** del paquete LibreOffice de Linux. Básicamente la fórmula establece las siguientes políticas:

* En la zona la cual va a ingresar un vehículo a una hora específica sus plazas libres en ese momento no pueden ser menores que cero.
* En la zona la cual va a salir un vehículo a una hora específica sus plazas libres en ese momento no pueden ser mayor a las plazas totales inicializadas en dicha zona.

Después de implementar una fórmula de aleatoriedad en Office Calc para la distribución de la cantidad de carros en distintas zonas, se obtiene un libro de cálculo en Office Calc con cuatro hojas, ver Anexo C Tabla completa de datos. La primera hoja establece la condición del sistema a probar como se muestra en la figura 43.

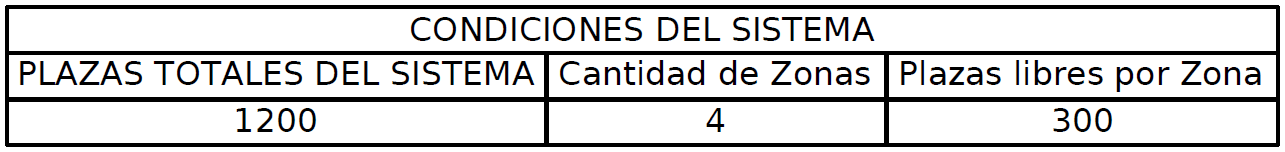


Figura 43. Condiciones del sistema para realizar las pruebas en TTCN-3

A pesar que el diseño del sistema considera tres controladores de zonas como máximo, las cuales pueden tener hasta 3 zonas con una capacidad máxima de 300 plazas libres y totales; las pruebas en TTCN-3 se ha hecho pruebas a dos controladores y sobre dos zonas específicas sobre dichos controladores. Lo anterior se hace por fines prácticos pero no implica que las pruebas a realizar no sirvan como técnica para la minimización de errores en el diseño.

La segunda y tercera hoja llamada ***ENTRADA* y *SALIDA*** se encuentran la cantidad de vehículos que han ingresado o salido del sistema con su respectiva distribución a cada una de las zonas, La figura 44 muestra una fracción de los vehículos que ingresaron al sistema hasta las 7:25 a.m. y la figura 45 muestra una fracción de la distribución por zona de los vehículos que ingresaron al sistema hasta las 7:25 a.m.

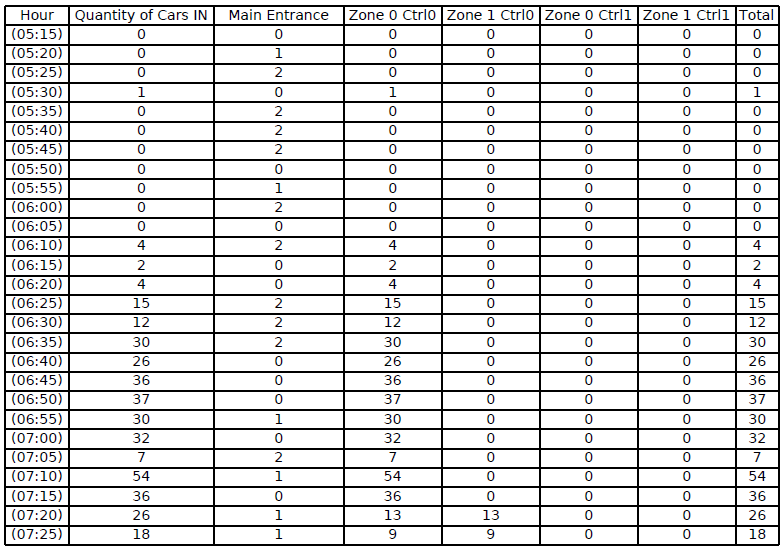


Figura 44. Distribución de carros ingresados al sistema y a cuatro diferentes zonas.

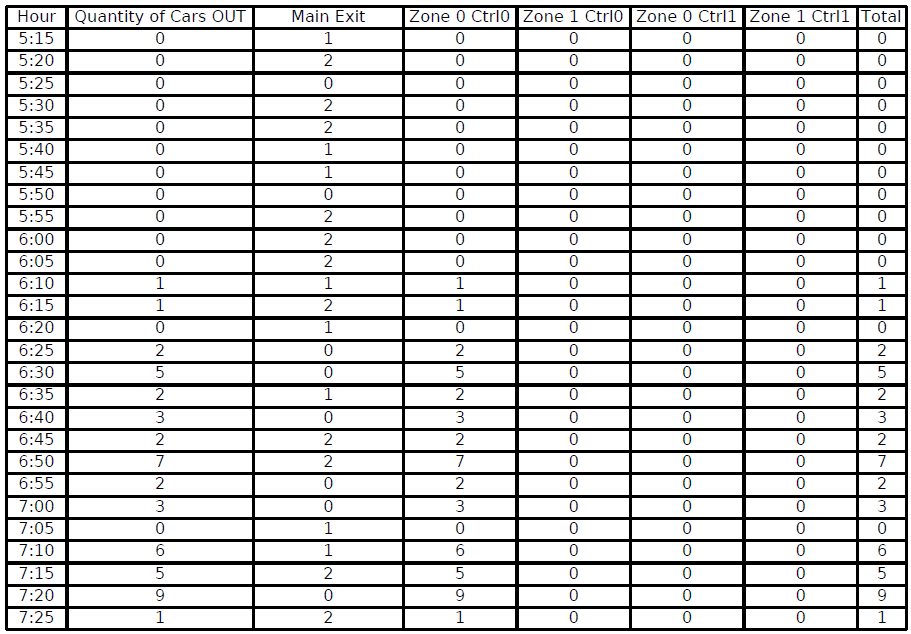


Figura 45. Distribución de carros que salieron del sistema y de cuatro zonas.

Finalmente el libro de cálculo en Office Calc posee una hoja donde se presenta el consolidado de plazas libres por zona y por hora. Como se muestra en la figura 46, la última columna sirve para identificar una hora en especial, por ejemplo: Si queremos ver las plazas libres que hay en cada zona de parqueo a las 7:10 a.m. es equivalente al ver la fila del índice que tiene el valor de 23.

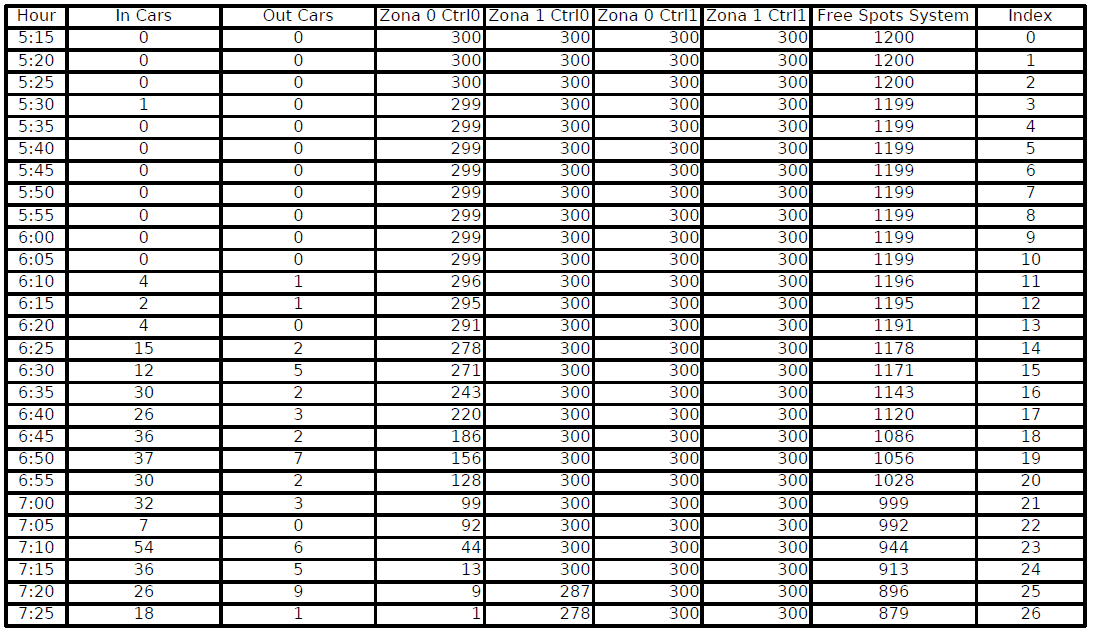


Figura 46. Consolidado de plazas libres por zona y en el sistema en general de parqueo.

Una vez distribuido la información de vehículos que han ingresado y salido del sistema y haber estructurado el consolidado de plazas libres por zonas y a nivel del sistema. Se traduce la información anterior en un tipo de dato que pueda ser leído por el lenguaje de TTCN-3.

El lenguaje de TTCN-3 posee dentro de su arquitectura la posibilidad de implementar adaptadores para estimular el sistema bajo prueba, dichos adaptadores se diseñan por medio de lenguajes de programación como C o C++. La herramienta RTDS hasta el momento no soporta todo el lenguaje de TTCN-3, para más información de las limitaciones del lenguaje TTCN-3 en RTDS véase el manual de referencia de Pragmadev. Por ende, se ha hecho uso de vectores unidimensionales que representan la información de entrada y salida de vehículos y su respectivo valor esperado por plazas libres.

Se implementa un algoritmo en el lenguaje de programación Python para la transformación de una tabla en formato \*.xls a un vector en el lenguaje de TTCN-3. La figura 47, muestra la función principal del algoritmo.

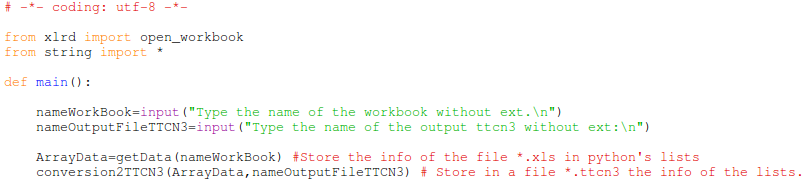


Figura 47. Función principal del algoritmo para transformar datos de un archivo \*.xls a un vector válido en TTCN-3.

Básicamente la función principal solicita al ingeniero de pruebas que ingrese el nombre del archivo de formato \*.xls. El formato que debe de tener es basado en las figuras 44,45 y 46, de lo contrario la información generada por el algoritmo no será coherente. El nombre del archivo debe ser escrito sin especificar el formato. Adicionalmente el ingeniero de pruebas puede establecer el nombre que le dará a su archivo \*.ttcn3, lo anterior es porque en el lenguaje TTCN-3 el nombre del módulo debe de coincidir con el nombre de dicho archivo.

Una vez que el ingeniero de pruebas haya ingresado los datos requeridos, el algoritmo hace llamado a dos funciones que son: ***getData*** y ***conversion2TTCN3***. La primera función retorna una lista de Python equivalente con los datos del archivo \*.xls; posteriormente esta lista generada es un parámetro de la función ***conversion2TTCN3,*** dicha función se encarga de convertir cada término de la lista de Python en vectores unidimensionales válidos en TTCN-3. Para ver todo el código del algoritmo mencionado previamente, véase el Anexo C.

Una vez obtenido los datos que se van a inyectar en el sistema bajo pruebas, se diseña la estructura de la prueba en TTCN-3 sobre la herramienta RTDS. Los siguientes pasos son recomendaciones para diseñar una prueba en TTCN-3 de forma sencilla.

1. Defina cuatro módulos de TTCN-3.
2. El primer módulo hará referencia solo a los tipos de datos que se usarán en las pruebas.
3. El segundo módulo corresponderá a los mensajes, ***templates***, que se enviarán al sistema bajo prueba.
4. El tercer módulo se definirán los puertos y componentes donde se compartirán los mensajes.
5. El cuarto módulo será el banco de pruebas. Aquí estarán los casos de prueba que se desea realizar además de la definición del control.

La figura 48 ilustra el primer módulo que hacer referencia a la definición de los tipos de datos a usar en las pruebas.



Figura 48. Declaración de los tipos de datos y señales que se usarán en las pruebas de TTCN-3.

La figura 48, enmarca un tipo singular de definir las señales que van a ser usadas sobre el sistema bajo prueba, dado que el nombre de éstas corresponden tal cual a las definidas en el diseño del sistema, lo anterior es una regla de la herramienta RTDS para la construcción de pruebas en TTCN-3. Adicionalmente se estipula que para las señales que no tiene parámetros se pueden definir como tipos de datos ***enumerated***  mientras que las señales que si tienen se debe de definir como tipos de datos ***record***.

La forma de declaración de señales en TTCN-3 se asemeja a los lenguajes de programación C, C++, Java entre otros; donde primero se define el tipo de dato de la variable y seguido del nombre de ésta.

La figura 49. De definen las plantillas, ***templates***, del diseño de la prueba.

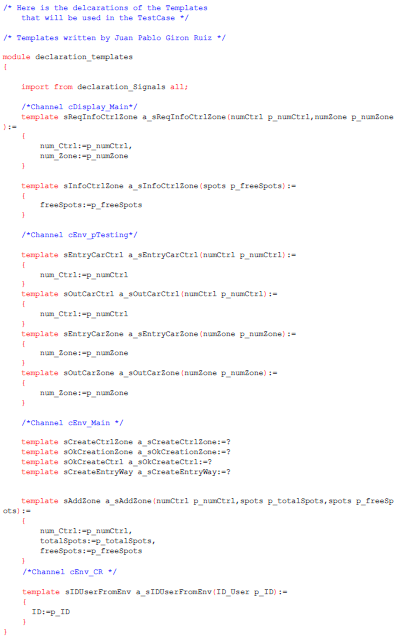


Figura 49. Definición de templates a ser usados en la prueba en TTCN-3.

La razón por la cual se crean templates en TTCN-3 es porque estos son los que se enviarán al sistema bajo prueba; además tienen la propiedad que los parámetros de la señal sean definidos varias veces, lo que permite enviar al sistema bajo prueba una señal con distintos datos; caso contrario como se hacía con la interfaz gráfica que solo se podía modificar un parámetro de una señal por botón. Nuevamente existe una diferencia de declaración de templates cuando las señales tienen parámetros a las que no tienen. El símbolo ? en el lenguaje TTCN-3 quiere indicar que cuando se reciba o se envíe una señal sin parámetros no interesa su contenido. Las señales del diseño en SDL que contengan parámetros son definidas en los templates indicando el nombre de cada parámetro y su respectivo valor.

La Figura 50, define los puertos y el componente el cual se va a realizar la prueba.

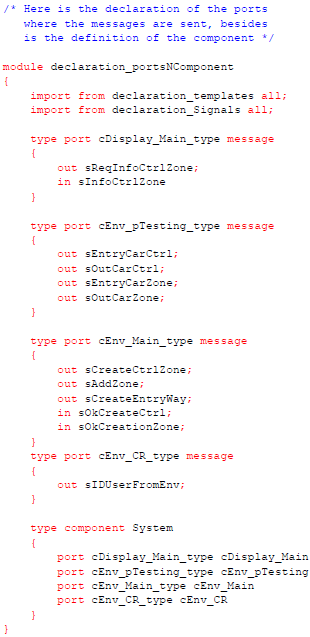


Figura 50. Definición de los puertos y componentes de la prueba en TTCN-3

En la figura 50, se define la estructura del módulo de prueba; los puertos en TTCN-3 son equivalentes a los canales en SDL. El nombre que se define para el puerto debe ser el mismo que el nombre definido en el canal del diseño en SDL. El componente en TTCN-3 indica el sistema en SDL que va a hacer probado.

Hasta el momento se ha definido las estructuras básicas para definir casos de prueba al diseño del sistema en SDL usando TTCN-3. La figura 51 representa la definición de un caso de prueba para probar a partir de un escenario de entrada y salida de vehículos que a una hora determinada la cantidad de plazas libres del sistema de parqueo y de cada una de las zonas corresponden al valor esperado.



Figura 51. Construcción de un banco de pruebas en TTCN-3.

Como se observa en la Figura 51, se ha diseñado una serie funciones y casos de prueba con el objetivo de determinar las plazas libres de una zona de un controlador específico a una hora determinada. Cuando se desea estimular el sistema con una señal se hace a través de la sentencia ***send***; por el contrario si se espera un señal se usa la sentencia ***receive*** que es de bloqueo, lo que significa que la prueba no avanza hasta que dicha señal haya llegado. El lenguaje de TTCN-3 cuenta con mecanismo de coincidencia que permite evaluar si el valor de la señal recibida corresponde con el valor esperado, de ser así el caso de prueba, ***testcase,*** retorna un valor de ***pass***; de lo contrario retornará ***fail*** si los valores no coinciden o ***error*** si existe alguna falla en el diseño de la prueba en TTCN-3.

En la parte de control del módulo del diseño de la prueba se especifica que ***testcase*** va a ser ejecutado; si es más de un caso de prueba, en la parte del control se da un veredicto total con base a cada uno de los veredictos de las pruebas, cuyo valor es ***pass*** si y solo si cada uno de los resultados de los casos de prueba son del mismo valor; de lo contrario el resultado total será ***fail.***

Las funciones que se definieron en la figura 51 corresponden al ingreso o salida de un vehículo de una zona en un controlador específico, además de una función encargada de enviar un requerimiento de información de cuantas plazas tiene libre una zona de un controlador específico. Los casos de prueba que se definieron son los siguientes:

|  |  |
| --- | --- |
| Nombre del caso de prueba (***test case***) | Propósito/Resultado |
| tc\_initialization | Esperar que el sistema cumpla la primera etapa de inicialización antes de enviar los estímulos. El resultado de esta prueba siempre será ***pass.*** |
| tc\_creationCtrl | Crea un controlador en el sistema de parqueo. El resultado de esta prueba siempre será ***pass*** si después de la creación se recibe la señal a\_sOkCreateCtrl. |
| tc\_CreationZone | Crea una zona en un controlador específico, ajustando su capacidad siempre a 300 plazas libres y totales. El resultado de éste caso de prueba será ***pass*** cuando se reciba la señal a\_sOkCreationZone. |
| tc\_EntryCar | Simula a través del proceso pTesting que un carro ha ingresado a una zona de un controlador específico. El resultado siempre será ***pass.*** |
| tc\_OutCar | Simula a través del proceso pTesting que un carro ha salido de una zona de un controlador específico. El resultado siempre será ***pass.*** |
| tc\_VerifyFreeSpots | Solicita al sistema la cantidad de plazas libres de una zona en un controlador específico. Estipula cuando debería de ser el valor esperado. El resultado de este caso de prueba será ***pass*** si la cantidad de plazas totales que tiene dicha zona corresponde al esperado. |

Tabla 6. Definición de las funciones y casos de prueba construida en TTCN-3

Nuevamente en el banco de pruebas se declaró una variable llamada ***indexHour,***  dicha variable tiene como funcionalidad especificar el índice correspondiente a la hora como se explicó anteriormente en la tabla de datos construida.

Las figuras 52 y 53, muestran el resultado de la prueba al esperar que las plazas libres de la zona 0 del controlador 0 del sistema sea de 156.

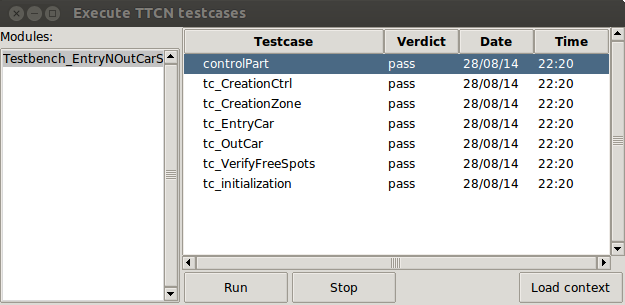


Figura 52. Resultado de la prueba de verificar las plazas libres de una zona en un controlador específico a una determinada hora.

EN LA FIG

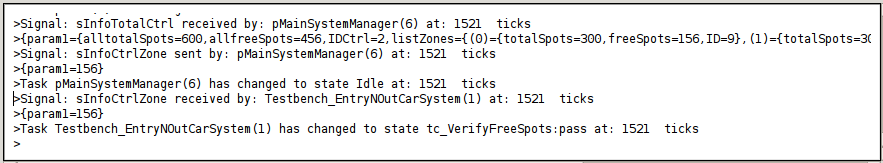


Figura 53. Reporte de la última señal con el valor esperada de la cantidad de plazas libres en la zona 0 del controlador 0 en la herramienta RTDS.

Después de la fase de creación de controladores y zonas a través de TTCN-3, las plazas libres y totales de todas las zonas de los controladores son 1200. Estimulando el sistema con entrada y salidas de vehículos, véase la figura 44 y 45; a las 6:50 a.m. que corresponde al índice 19, según la figura 46 se espera que la zona 0 del controlador 0 tenga 156 plazas libres. Efectivamente el diseño del sistema funcionó satisfactoriamente.

## Verificación Formal usando IFx

1. Una zona es un área donde se pueden aparcar vehículos y cuenta con un sistema de sensores para determinar cuando entran o salen los vehículos. [↑](#footnote-ref-1)
2. La autogestión del sistema se refiere a que el sistema de parqueo puede pedir información a los controladores de zonas de forma independiente. [↑](#footnote-ref-2)