WS-POS

V 1.3.1

Change bar version

OMG Document Number: formal/20-01-04

Normative Reference: https://www.omg.org/spec/WS-POS/1.3.1

Associated Normative Machine Consumable Files:

https://www.omg.org/spec/WS-POS/20180301/JAX-WSContract.zip
https://www.omg.org/spec/WS-POS/20180301/WCFContract.zip
https://www.omg.org/spec/WS-POS/20180301/WSDL.zip
https://www.omg.org/spec/WS-POS/20180301/XSD.zip

Copyright © 2008, National Retail Federation
Copyright © 2022, Object Management Group, Inc.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.
Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 9C Medway Road, PMB 274, Milford, MA 01757, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see: http://www.omg.org/legal/tm_list.htm. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/report_issue.)

Chairman UnifiedPOS Committee:

H Paul Gay	Epson America

UnifiedPOS Committee Members:

Gerald Armentrout	IBM
Kazunori Chihara	Seiko Epson Corporation
Kunio Fukuchi	Fujitsu Frontech Limited
Tadashi Furuhata	Seiko Epson Corporation
Denis Kuniss	Wincor-Nixdorf
Jürgen Moser	Bizerba
Lawrence Owen	STAR MICRONICS CO., LTD.
Daniel Schwertführer	Bizerba
Brian Spohn	NCR Corporation
Michael Webb	Data Logic, Inc.

Contributors:

Richard Halter	Global Retail Technology Advisors, LLC

OPOS-J Work Team For WS-POS Version 1.3.1

Chairmen OPOS-J Work Team:

Toru Yanagisawa	NEC Platforms, Ltd.
Masanori Sambe	Toshiba TEC Corporation
Eiki Murakami	Sorimachi Giken Co., Ltd.

Core OPOS-J Members:

Tadashi Furuhata	Seiko Epson Corporation
Hideo Nakamura	Seiko Epson Corporation
Kunio Fukuchi	Fujitsu Frontech Limited
Toyohiro Yasumoto	VINX CORP.
Tako Tamura	Sorimachi Giken Co., Ltd.
Kiyotaka Abe	Sorimachi Giken Co., Ltd.
Kenichi Nagai	STAR MICRONICS CO., LTD.
Yuji Mori	STAR MICRONICS CO., LTD.
Akio Tajima	NCR Japan, Ltd.
Takahiro Akutsu	Hitachi Information & Communication Engineering, Ltd.
Mitsuhiro Igarashi	NEC Platforms, Ltd.
Takahide Kubota	Toshiba TEC Corporation
Kazunori Chihara	Seiko Epson Corporation
Toshihiko Hayashi	OMRON SOFTWARE Co., Ltd.

Contributors OPOS-J:

Soichi Fujii	Microsoft Co., Ltd.
Hiroshi Ota	Microsoft Co., Ltd.
Mitsuo Nagata	VINX CORP.

TABLE OF CONTENTS

1. PREFACE 7

1. ABSTRACT 8

1.1 OVERVIEW *UPDATED IN VERSION 1.3* 8
1.2 CONFORMANCE REQUIREMENTS 9
1.3 ARTS WS-POS STANDARDS STACK 10
1.3.1 WEB SERVICE COMPONENTS 11
1.3.2 ADDITIONAL COMPONENTS *UPDATED IN VERSION 1.2* 11
1.4 OUT OF SCOPE *UPDATED IN VERSION 1.2* 11

2. WS-POS COMPONENTS 12

2.1 EXPLANATION OF WS-POS RELATED TERMINOLOGY 12
2.2 MESSAGING BASE 13
2.3 SERVICE DESCRIPTION AND DISCOVERY 14
2.4 WS-POS BEHAVIOR MODELS *ADDED IN VERSION 1.2* 18
2.4.1 INTRODUCTION TO PROPERTIES, METHODS AND EVENTS *ADDED IN VERSION 1.2* 18
2.4.1.1 PROPERTY, METHOD AND EVENT *ADDED IN VERSION 1.3* 19
2.4.2 WS-POS COMMUNICATION MODEL *ADDED IN VERSION 1.2* 19
2.4.3 SESSION MANAGEMENT AND DEVICE CONTROL *ADDED IN VERSION 1.2* 20
2.4.4 INTRODUCTION OF WS-POS SESSION MANAGER CONCEPT *ADDED IN VERSION 1.2* 20
2.4.5 IDENTIFYING WS-POS SESSION *ADDED IN VERSION 1.2* 21
2.4.5.1 SECURITY CONSIDERATION *ADDED IN VERSION 1.3* 21
2.4.6 TYPICAL SEQUENCE TO ESTABLISH WS-POS SESSION *ADDED IN VERSION 1.2* 22
2.4.7 CALLING WS-POS SERVICE METHOD SAND USING PROPERTIES *ADDED IN VERSION 1.2* 23
2.4.8 MULTIPLE WS-POS SERVICE CONSUMER CLAIM REQUESTS ON A WS-POS SERVICE PROVIDER *UPDATED IN VERSION 1.2* 24
2.4.9 WS-POS METHOD SAND DEVICE METHODS *UPDATED IN VERSION 1.3* 29
2.4.10 WS-POS EVENTS HANDLING USING BI-DIRECTIONAL COMMUNICATION *UPDATED IN VERSION 1.2* 30
2.4.11 WS-POS VENTS HANDLINGON POLLING *ADDED IN VERSION 1.2* 33
2.4.12 RESOLUTION OF FREQUENT COMMUNICATION EVENTS IN THE EVENT NOTIFICATION *ADDED IN VERSION 1.3* 38
2.4.13 WS-POS SERVICE NETWORK CONNECTION MANAGEMENT CONSIDERATIONS *ADDED IN VERSION 1.2* 43
2.4.14 WS-POS SERVICE NETWORK CONNECTION MANAGEMENT, EVENT – BI-DIRECTIONAL COMMUNICATIONS *ADDED IN VERSION 1.2* 46
2.4.15 WS-POS SERVICE NETWORK CONNECTION MANAGEMENT, EVENT – POLLING *ADDED IN VERSION 1.2* 51
2.4.16 WS-POS METHOD REFERENCES (UPOS UML STYLE) *UPDATED IN VERSION 1.2* 58
2.4.17 WSPOS EVENT AND WSPOS EVENT RESPONSE *ADDED IN VERSION 1.2* 70
2.4.18 WS-POS EVENT REFERENCE IN BI-DIRECTIONAL COMMUNICATION *UPDATED IN VERSION1.3* 76
2.4.19 MODIFICATIONS TO XMLPOS *UPDATED IN VERSION 1.3* 82
2.4.20 FILE PATH FOR A METHOD CALL *ADDED IN VERSION 1.2* 87
2.4.21 RETRIEVING CONSUMERID THAT HAS BEEN SUCCESSFULLY CLAIMED *ADDED IN VERSION 1.3* 88
2.5 STATUS, STATE MODEL AND EXCEPTIONS 92
2.6 SHARED DEVICE MODEL 93
2.7 EVENT MESSAGES 94
2.8 INPUT MODEL 96
2.9 OUTPUT MODEL 98
2.10 DEVICE POWER REPORTING MODEL 100
2.11 ARTS XMLPOS COMMAND SET 101
2.12 ARTS XMLPOS EVENT SET 101

2.13 ARTS XMLPOS SCHEMA 102
2.14 WS-POS WSDL 103
2.15 BACKWARD COMPATIBILITY *ADDED IN VERSION 1.2* 108
2.16 SECURITY 108
2.17 XML PAYLOAD 110

3. GENERAL FLOW 111

3.1 GENERAL WS-POS FLOWS 111
3.2 SIMPLE USE CASE 112
3.3 USE CASE CATALOG 116
3.4 SCOPE 116
3.5 SUB SCOPE: LINKAGE BETWEEN IN-STORE KIOSK AND POS (DEVICES) 120
3.6 SUB SCOPE: LINKAGE BETWEEN SALES ASSISTANCE TERMINALS AND POS DEVICES 124
3.7 SUB SCOPE: BATCH PAYMENT IN COMMERCIAL COMPLEX 129
3.8 SUB SCOPE: MONITORING IN-STORE KIOSK EQUIPMENT AND COOPERATION WITH BACK-OFFICE ON OCCURRENCE OF PROBLEMS. 133
3.9 SUB SCOPE: POS SYSTEM IN CONSIDERATION OF COOPERATION BETWEEN VARIOUS INDUSTRIES 138
3.10 SUB SCOPE: LINKAGE BETWEEN DISPLAY SHELF AND REAR SYSTEM 142
3.11 SUB SCOPE: COOPERATION BETWEEN ELECTRONIC SHELF LABEL AND SHELVING ALLOCATION INFORMATION 150
3.12 SUB SCOPE: SELF-SERVICE REFUELING 155

4. DOCUMENT HISTORY 163

5. REFERENCED DOCUMENTS AND SOFTWARE SUPPORT FILES 166

5.1 REFERENCED DOCUMENTS 166
5.2 SOFTWARE SUPPORT FILES 166
5.3 SOFTWARE SUPPORT FILES *ADDED IN VERSION 1.2* 168
5.4 SOFTWARE SUPPORT FILES *ADDED IN VERSION 1.3* 168

6. WS-POS CLASS DIAGRAMS *UPDATED IN VERSION 1.2* 170

7. APPLICATION DEVELOPMENT SUPPORT *UPDATED IN VERSION 1.2* 171

1. Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

http://www.omg.org/spec

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
9C Medway Road, PMB 274
Milford, MA 01757
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to https://issues.omg.org/issues/create-new-issue

1. ABSTRACT

1.1 Overview *Updated in Version 1.3*

The W3C Glossary, http://www.w3.org/TR/ws-gloss/, defines a Web Service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-understandable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Web Services for Point of Service (WS-POS) is a technical document intended to provide retail devices, terminals and servers with the capabilities necessary to interoperate in a detached, dynamic network as well as more typical retail LANs by leveraging these W3C specifications.

WS-POS Version 1.1 represents an update of the original release of the standard with emphasis on refined WSDL support for all UnifiedPOS Version 1.14 peripheral devices as well as enhancements and bug fixes. This amended material was created almost exclusively as a result of implementation experience and subsequent refinement submissions from the OpenPOS Technology Council of Japan.

WS-POS Version 1.2 builds on the previous versions and adds important enhancements to provide better Event management, adding "heart beat" methodology to aide in application interaction with network attached POS peripherals, provide for "polling" of POS peripherals to help with Event services, and adding logic for a consumer ID to aide in management of services available to an application program.

In WS-POS version 1.3, version 1.2 as the base, taking into account for the transmission and reception of binary data, how to resolve frequent event communication in the event notification for more efficient handling of those events with embedded parameter returns with event notification, and how to acquire the encrypted value of the consumer ID which has been successful in Claim for the use case where multiple WS-POS service consumers shares one WS-POS service provider and one of the consumer unintentionally locks the service provider have been added.

A significant part of this document is a profile of the minimal web-service specifications necessary to support remote device interoperability. This profile serves to constrain, articulate, and enable the usage of the WS specifications in order to facilitate interoperability and ensure appropriateness of the chosen web-services specifications for a Web Services implementation at the retail store. The WS-POS specification also utilizes UnifiedPOS XMLPOS and other significant details intended to permit an easy and interoperable implementation of a Web Service for Point of Service solution.

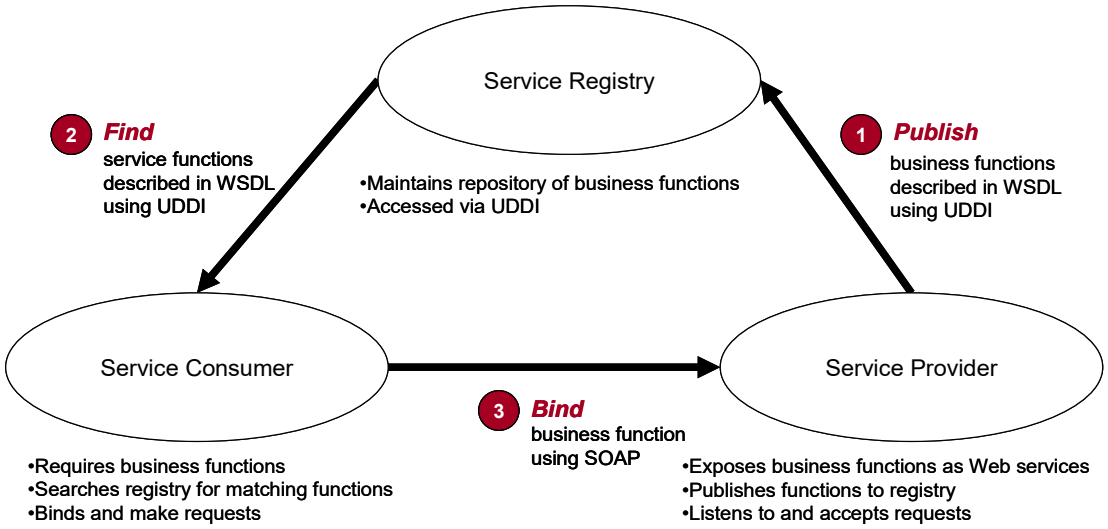


Figure 1: Web Services Architecture

Figure 1 illustrates a typical connectivity model for Web Services in general, and more importantly, the one that will be used for WS-POS. First, a Service Provider registers with a Service Registry (UDDI) identifying the services it provides. Second, the Service Consumer finds the locations of the Service Providers who supply the services it needs to perform its task. Finally, it binds directly to the identified Service Providers to exchange the necessary messages to perform that service.

1.2 Conformance Requirements

In order to ensure interoperability as defined by this specification, an implementation must meet all the requirements set forth in this specification. These requirements are defined using the following format:

Id	Name	Description
XY001	Requirement Name	This is a requirement description

IP POLICY

This specification was originally created under the ARTS IP Policy which can be found here: http://www.omg.org/cgi-bin/doc?retail/2017-12-01. With the transition to the Object Management Group, this standard is now published under a default reasonable and non-discriminatory ("RAND") licensing obligation for members with only limited exceptions.

Note: The following XML examples include "namespace references". These are not actual file locations but placeholders for the appropriate namespace where the support files can be found.
In summary, when an application uses the schema examples as a basis for their code, it is necessary to replace the placeholders with valid namespace locations.

1.3 WS-POS Standards Stack

The following diagram shows the WS-POS stack components necessary to be compliant with this standard.

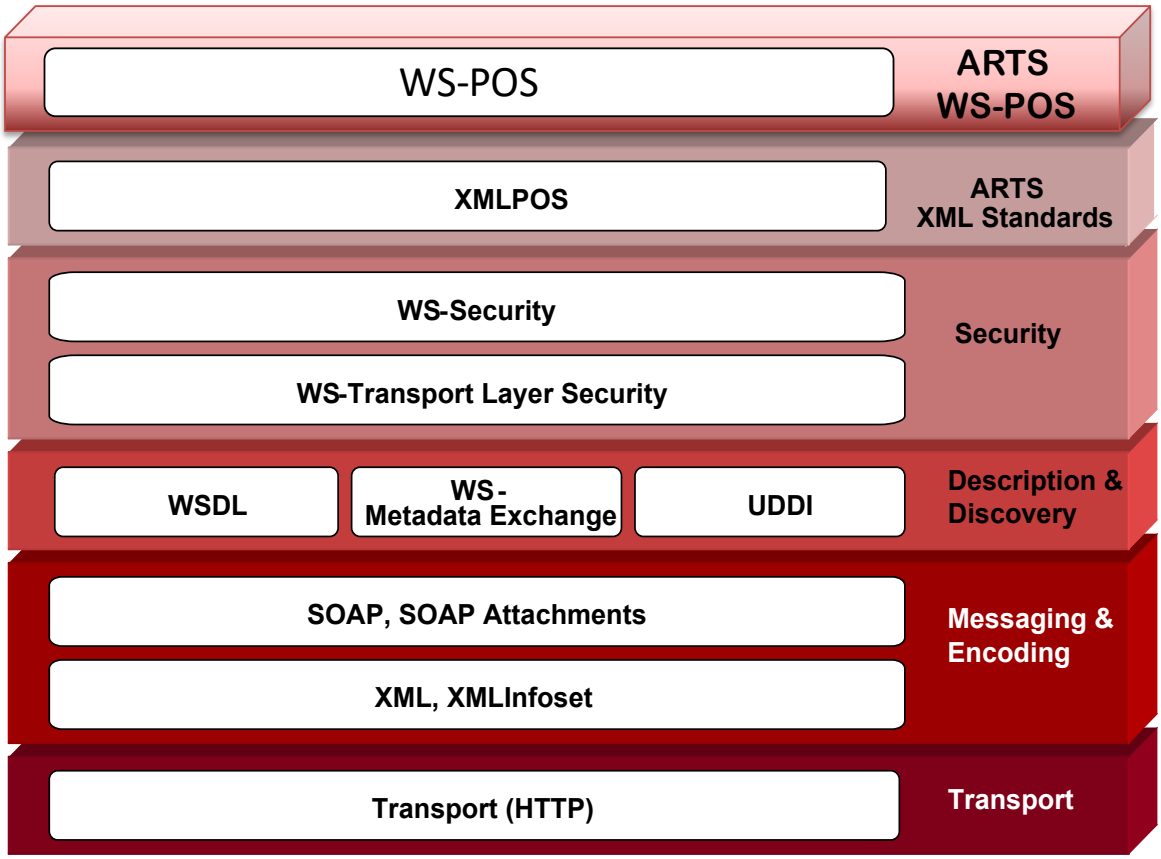


Figure 2: WS-POS **Stack**

1.3.1 Web Service Components

The web services definition of WS-POS addresses these key areas:

- The Messaging base: a description of the core that all messaging in WS-POS is based upon.
- Service discovery: How resources and services are located.
- Service description: How services and messages of interest will describe themselves.

1.3.2 Additional Components *Updated in Version 1.2*

- Security: provides a description of the mechanisms that must be in place to ensure an appropriate level of security.
- XML Payload: The payload of a message.
- Service Definitions: In order to enable clients to easily find available devices using Universal Data Discovery and Integration (UDDI), it is necessary to define the names and definitions of device services.
- ***(Added in Version 1.2)*** Methods for handling unintended disconnect scenarios between a WS-POS Service Provider and a WS-POS Service Consumer (see section 2.4 WS-POS Behavior Models).

1.4 Out of Scope *Updated in Version 1.2*

Management, dynamic service discovery, and events at the web service level have not been addressed in this version of the specification.

This version limits itself to data flow within the "retail store" environment. Future versions may expand in scope to include data flow to and from the enterprise. This has the implication that all security outside of interactions between UDDI, Client, and Device require implementation practices considered under the guidance of good systems design practices.

This specification does not specify:

- Methods for selecting a device service to use from a list of possible devices services. This is an area an application needs to manage.
- Methods to preload devices with security credentials or validate security credentials.
- How to verify that devices or clients are valid.
- Address Payment Card Industry (PCI) data requirements. It is recognized that the information contained within the messages will need to follow the PCI standard. Those requirements are defined by the PCI committee.

2. WS-POS COMPONENTS

This section provides details concerning each component of the WS-POS stack.

2.1 Explanation of WS-POS Related Terminology

2.1.1 XML POS

XML POS is a definition in XML of the operations, states, and attributes that should be supported by POS devices standardized by ARTS.

2.1.2 Web Service

A Web Service provides a standard method of interoperability for different software to operate on various platforms (W3C). Many of the implementations are defined in the WSDL definition language and use a protocol to exchange information in XML format called SOAP while using HTTP for transport. In some cases, Web Service refers to the technology itself while in other cases it refers to services implemented by the technology.

2.1.3 Web Service based on the WS-POS specification

This specification describes a Web Service based on the WS-POS specification as a WS-POS service.

2.1.4 Provider and consumer of a WS-POS service

This specification describes the WS-POS service provider who is providing the WS-POS service and WS-POS service consumer who is using the WS-POS service.

2.1.5 Methods in WS-POS

Methods in WS-POS are expressed as Web Service operations. This specification uses WS-POS methods instead of the term of Web Service operations.

2.1.6 Properties in WS-POS

Properties in WS-POS are realized using Getter/Setter methods. For example, the **DeviceEnabled** property defined by UnifidPOS is expressed by the **GetDeviceEnabled** method as a Getter method while the **SetDeviceEnabled** method is expressed as a Setter method. In read-only properties, there are no Setter methods. This specification typically does not describe Getter method/Setter methods hereafter. Please note that Getter/Setter methods are used to acquire/set properties.

2.1.7 Events in WS-POS

Events in WS-POS notify WS-POS service consumers about device events defined by Unified POS from WS-POS service providers.

WS-POS calls the client from the Web Service in order to receive device events defined in Unified POS by the application. Applications that receive events must notify the services about the end point for reception.

2.2 Messaging Base

2.2.1 Description

The messaging base is the set of technologies utilized as a core basis for all the other areas. It provides a starting point for description of all the messages that are employed.

2.2.2 WS-I Basic Profile

A WS-POS service is described by WSDL 1.1, based upon WS-I Basic Profile version 1.2. This consists of using SOAP 1.1 over HTTP, with WS-Addressing Core 1.0, and Web Service Description Language (WSDL) Version 1.1. The WSDL 1.1 description includes the service offered (type/interface), method of use (bind), and location (endpoint). These are the mainstream defaults for web service usage, and are typically seen as the "ground rules" by which the Web Services "world" operates.

Id	Name	Description
MB001	Basic Profile	An implementation MUST be conformant to WS-I Basic Profile version 1.2

2.3 Service Description and Discovery

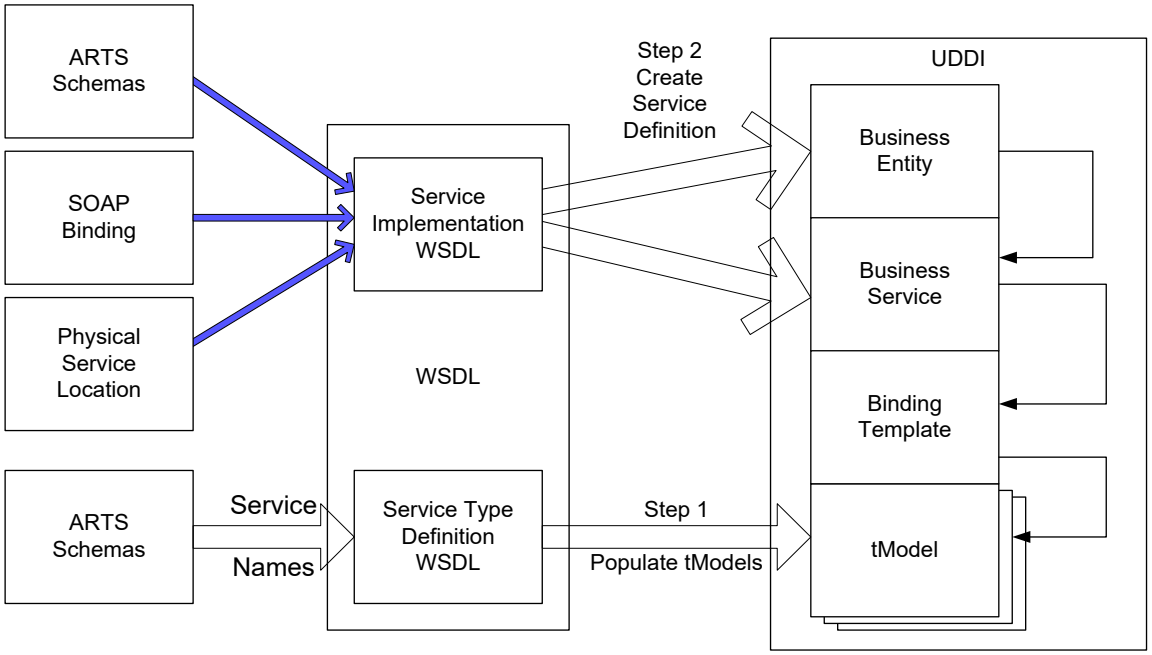


Figure 3: ARTS Schema - WSDL - UDDI Relationship

Populating a UDDI registry is a two-step operation. First, the tModel (Technical Model) is populated from a WSDL with the appropriate Service information, usually provided by a Service Provider. Then, the Business Service is populated with links to the group of tModels necessary to perform a specific Business Service.

From the registry a Service Consumer may then look up the Business Services it needs to perform its task. Once an appropriate Business Service is determined, the Service Consumer may query the UDDI registry for contact information for each individual Web Service of that type.

UDDI 3.0.2 can be used to discover WS-POS services. WS-POS service providers register WS-POS services in the UDDI service registry.

The relationships between the WSDL description and the information registered in UDDI are the noted as follows:

WSDL	UDDI
types message binding	tModel
Service	businessService
Port	bindingTemplate

In addition, the ARTS Device Type (Scanner, POSPrinter, etc.) is set in businessEntity of UDDI.

2.3.1 Service Description

2.3.1.1 Description

Standardized data is required to ensure interoperability by:

1. Formally describing the service being offered by the service provider
2. Defining the message structure necessary to invoke the functionality of the specific service from the service provider
3. Ensuring that the choreography of the service provider and the service consumer is in sync

An additional benefit that comes with a good service description is the ability to utilize effective commercial tools to assist in the development and verification of interoperability conformance between the service provider and the service consumer.

2.3.1.2 Metadata

WSDL 1.1, as explained in WS-I Basic Profile V1.2, is employed as the core service description facility.

WS-MetaDataExchange is required to retrieve the metadata information relevant to the respective Web services utilized.

2.3.2 Discovery *Updated in Version 1.2*

2.3.2.1 Description

The ability to find and work with peripheral devices on the LAN (or non-LAN devices represented by proxies on the LAN) is a key requirement for WS-POS. Thus, discovering existing services (or new services after they are introduced) is an important part of the overall solution.

2.3.2.2 Universal Description, Discovery, and Integration

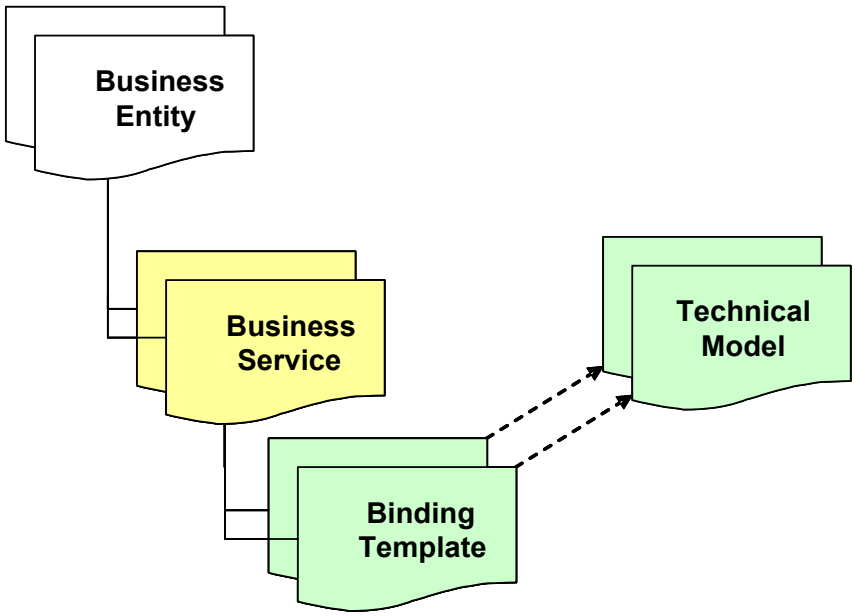


Figure 4: UDDI Overview

WS-POS uses version 3.0.2 of the Universal Description Discovery and Integration (UDDI) specification. A UDDI registry runs on a server and provides a single place for web services in a network to register, and to discover other web services. Version 3.x reduces much of the network chatter associated with previous versions of UDDI.

Id	Name	Description
DI001	UDDI	An implementation MUST be conformant to UDDI version 3.0.2

2.3.2.3 Registering a device service with UDDI

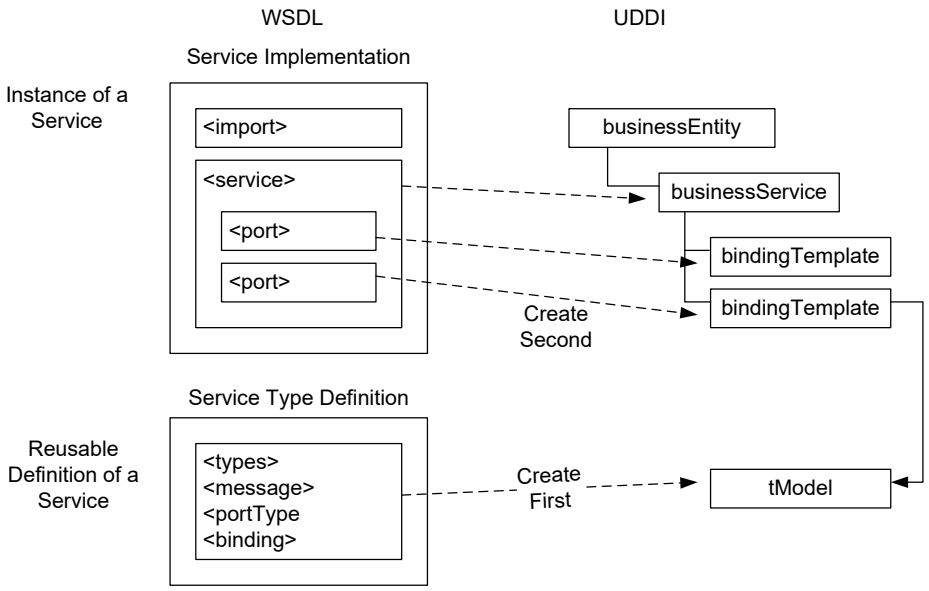


Figure 5: Registering a WSDL with a UDDI Registry

The device services register with UDDI using the mapping scheme described in the technical note from OASIS "Using WSDL in a UDDI Registry", Version 2.0.2.

2.3.2.4 Searching for a Service with UDDI

In order to locate a service, all that is required is a search of the UDDI for the business entities that contain the desired services as described in the table below.

ARTS Device Type (businessEntity)	User Service (businessService)	Administrative Service (businessService)
Scanner	ScanItems	ScannerAdmin

2.3.2.5 Removing a device service from UDDI

Device services should make every effort to ensure that they are not registered with UDDI before going out of service.

2.4 WS-POS Behavior Models *Added in Version 1.2*

The WS-POS specification defines cooperative behavior between a WS-POS service consumer that uses a WS-POS service and a WS-POS service provider who provides the service.

In WS-POS Version 1.2, the following behaviors were added.

Behavior	Background
Event polling methodology	A RIA (Rich Internet Application) that is running on the Web browser cannot open the port, since it is protected by the browser security program. The WS-POS Ver.1.1 specification required that the WS-POS service consumer open the event receiving end points. However, it is not possible for a RIA to implement this.
Network disconnection monitoring	WS-POS Ver. 1.1 was not able to facilitate good network connection management. There was no mechanism provided for either a WS-POS service consumer or a WS_POS provider to detect an interrupted network connection. This would lead to a fault situation where the WS-POS service provider could not release a claimed device.
Session management	In situations where multiple WS-POS service consumers required access to the same WS-POS service provider, WS-POS Ver. 1.1 did not provide for unique WS-POS consumer session identifiers. The WS-POS service provider could not identify and guarantee that it was responding to the correct WS-POS service consumer requestor.

2.4.1 Introduction to Properties, Methods and Events *Added in Version 1.2*

Over the network, WS-POS handles the Properties, Methods and Events that are described in Device Behavior Models in the ARTS UnifiedPOS standard. Refer to the “Device Behavior Model” in UnifiedPOS specification for more details.

2.4.1.1 Property, Method and Event *Added in Version 1.3*

In WS-POS, ARTSBinary of special encoded string type is used for property types which has binary data, method and event parameter type.

WS-POS service consumer must convert binary data which includes 0 to ARTSBinary to send it to WS-POS service provider.

Return value from WS-POS service provider may be ARTSBinary. Therefore, WS-POS service consumer must convert the return value.

The ARTSBinary is defined in XMLPOS. Refer Appendix A29-32 of UnifiedPOS 1.14.1 specification and ARTS XML Best Practices for the details of ARTSBinary.

2.4.2 WS-POS Communication Model *Added in Version 1.2*

WS-POS defines following two models for the communication between WS-POS service consumer and WS-POS service provider.

- Bi-Directional Communication
- Polling Communication *(Added in Version 1.2)*

To facilitate Bi-Directional Communication, the WS-POS service consumer opens a unique port for receiving events. This concept is
shown in Figure 6.

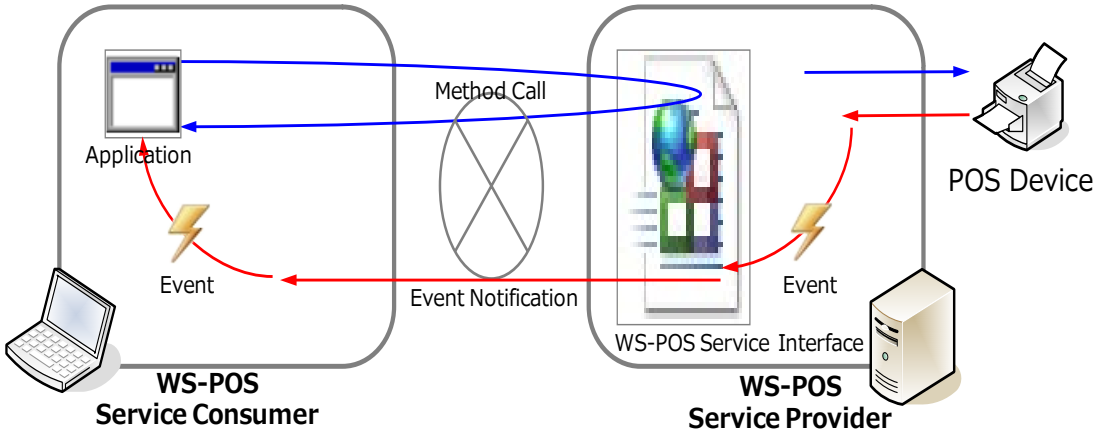


Figure 6 Bi-Directional Communication concept

Refer to Section 2.4.10 for more details on receiving events using the Bi-Directional Communication method.

WS-POS defines a Polling Communication mechanism whereby the WS-POS service consumer invokes a WS-POS service provider method call to enable the WS-POS service consumer to receive events. The concept is shown in Figure 7.

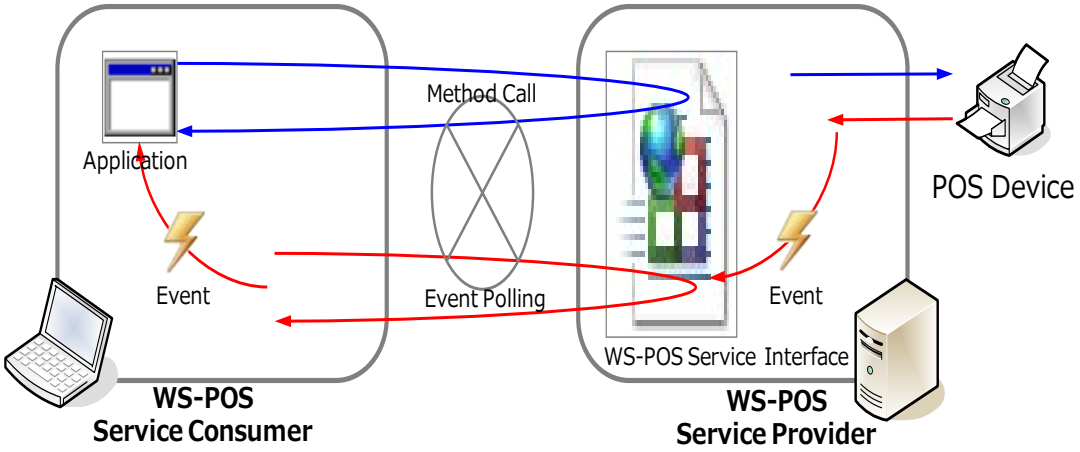


Figure 7 Polling Communication concept

Refer to Section 2.4.11 for more details on receiving events using the Polling Communication method.

2.4.3 Session Management and Device Control *Added in Version 1.2*

WS-POS handles the communication session between the WS-POS service consumer and the WS-POS service provider. It also performs the device control function for the application interfaces to the peripheral. Those two functions manage access and control at two different levels.

The WS-POS session management controls the connection between a WS-POS service consumer and a WS-POS service provider. Its primary functions include opening a session, closing a session and uniquely identifying the WS-POS service consumer where multiple WS-POS service consumers access a single WS-POS service provider.

The device control function of WS-POS service provider defines the application interfaces to access the UnifiedPOS device under its control. Its primary functions are opening (not opening a session), claiming, enabling, closing and allowing access to the Properties, Methods, and Events necessary to control a UnifiedPOS peripheral device.

Refer to Section 2.4.9 for more details on session management and using the device control API.

2.4.4 Introduction of WS-POS Session Manager Concept *Added in Version 1.2*

The WS-POS service provider must be able to respond to service requests from multiple WS-POS service consumers. WS-POS 1.2 defines the management of the network sessions between the WS-POS service provider and one or more WS-POS service consumers.

Successful session management is dependent upon a WS-POS service consumer having a unique ID to identify itself. The WS-POS service consumer injects its unique ID into the first message argument when the **openSession** method call is sent to the WS-POS service provider. Subsequent WS-POS service consumer and WS-POS service provider communications are managed as sessions identified by the unique consumer IDs.

The following diagram shows that the WS-POS service provider in each device has one session manager. Notice also that within the session manager, multiple uniquely identified instances of a

UnifiedPOS SO may exist. The session manager ensures that the WS-POS service consumer and the WS-POS service provider can successfully communicate with the correct SO Instance.

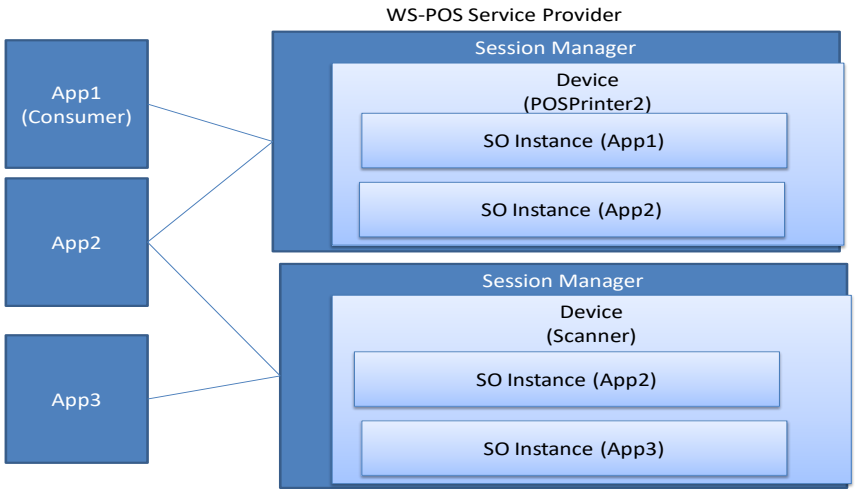


Figure 8: Session Management layer and Device Control layer

2.4.5 Identifying WS-POS Session *Added in Version 1.2*

The WS-POS service consumer issues its consumer ID but a standard method for generating a unique consumer ID should be utilized in order to ensure that it is unique within the system. (Usage of a GUID for Windows/.NET and a UUID for Java are recommended.)

If the WS-POS service consumer does not have a unique ID, the WS-POS 1.2 standard requires that a WS-POS service provider have the capability to create and unique ID and return it to the WS-POS service consumer. This request for a unique ID should only be necessary one time, since once the WS-POS service consumer then has a unique ID, it can store it and use it for all future sessions
with any WS-POS service provider.

It is important to note that it is mandatory that the WS-POS service consumer ID be unique and available before an open session request is made to the WS-POS service provider. This is necessary because the WS-POS service provider must have a WS-POS Consumer ID for an "**openSession**" method call and all subsequent P,M,E function requests the WS-POS service consumer and the WS-POS service provider.

2.4.5.1 Security consideration *Added in Version 1.3*

When the consumer ID of the WS-POS service consumer is available and readable by other WS-POS service consumers, a security breach of private information is possible. Only the originating consumer and the connected service provider should have access to the consumer ID.

For example:

Consumer (A) claims the service provider (B) MSR and a unique WS-POS consumer ID is used for identification. When a credit card is swiped and MSR TrackData is ready to be transmitted it should only be sent from B to A. If a different Consumer (C) has access to Consumer (A) unique ID, it

could be possible for Consumer (C) to imitate Consumer (A) and gain access to the MSR service (B) MSR TrackData.

To reduce this risk, each time an openSession method is called, a unique Consumer ID is dynamically generated and is only valid for that specific session.

For further information on consumer ID and security, see "2.4.21.1 Security Considerations".

2.4.6 Typical sequence to establish WS-POS session *Added in Version 1.2*

The following sequence diagram illustrates a typical WS-POS session establishment. In the diagram, a session is established for two WS-POS consumers. Then the respective UnifiedPOS devices are instantiated.

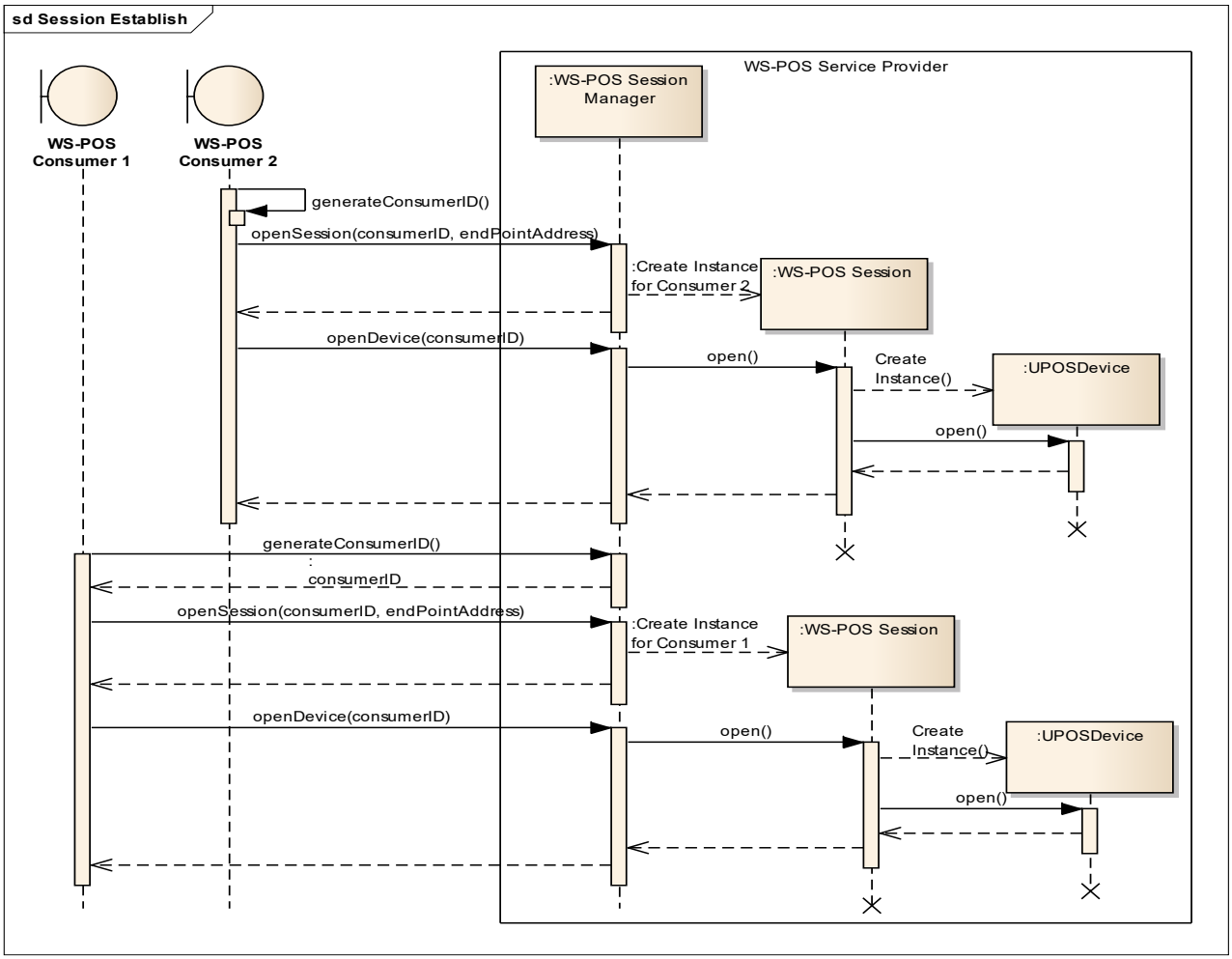


Figure 9: Establish Session

Note: The Consumer ID is a unique GUID or UUID

In the example above, the WS-POS Consumer 1 needs a unique ID. Using the endpoint address of the WS-POS service provider it invokes a **generateConsumerID** request to the WS-POS Session Manager. The unique ID is returned back to the WS-POS service consumer to use for all subsequent communication.

WS-POS Consumer 2 already has a unique Consumer ID or is capable of generating one by itself.

The following sequence for a POSPrinter **PrintNormal** method call illustrates the session management process.

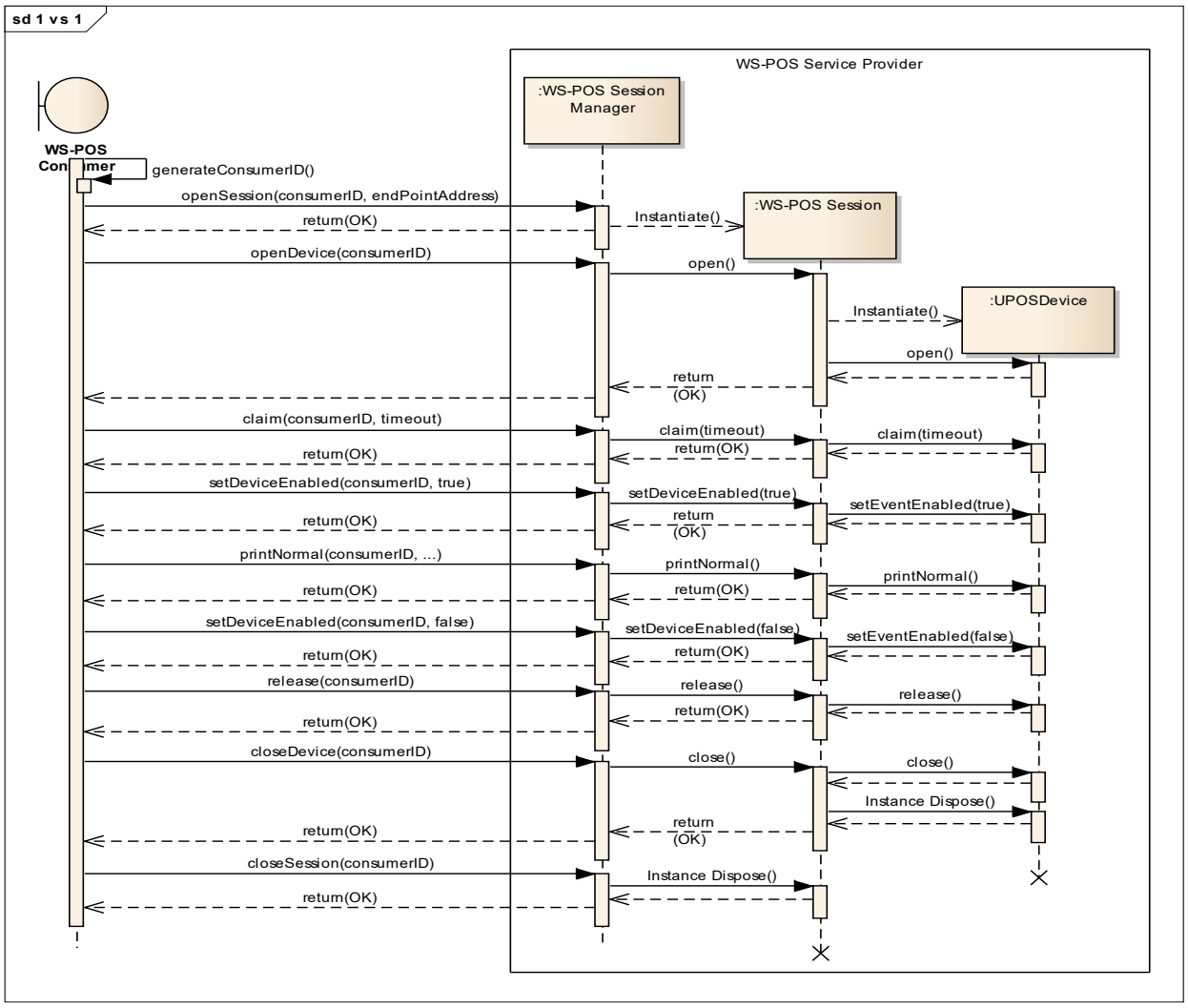


Figure 10: POS Printer - Print Normal example

Note that the Consumer ID is used by the Session Manager of the WS-POS service provide and not used by a UnifiedPOS device. A session is started using an **openSession** method and terminated using a **closeSession** method. The UnifiedPOS device driver is instantiated in **openDevice** method and it is terminated in the **closeDevice** method.

2.4.7 Calling WS-POS Service Methods and Using Properties

Added in Version 1.2

In order to enable a WS-POS service consumer to use the methods and the properties of a WS-POS service provider, the end point URL of the WS-POS service provider must be exchanged. The WSDL is used to ensure access to the necessary WS-POS service provider methods and properties.

For example, in modern programming, located in the WCF of .NET and the JAX-WS of Java, the proxy code for the WS-POS service is generated from the WSDL. The schema and the WS-POS service consumer use the WS-POS service provider functionality via this proxy code.

2.4.8 Multiple WS-POS Service Consumer Claim Requests on a WS-POS Service Provider *Updated in Version 1.2*

Once the WS-POS service consumer has initiated a successful **openDevice** method call to the enterprise WS-POS service provider, it must ensure that it has the focus to use the UnifiedPOS supported service provider functions. The WS-POS service consumer must issue a successful **claim** method call to the WS-POS service provider. Then it may be necessary to issue an **enable** method call before it can use the functions of the service provider.

The following sequence diagram shows the process of obtaining the claimed status of the UnifiedPOS device.

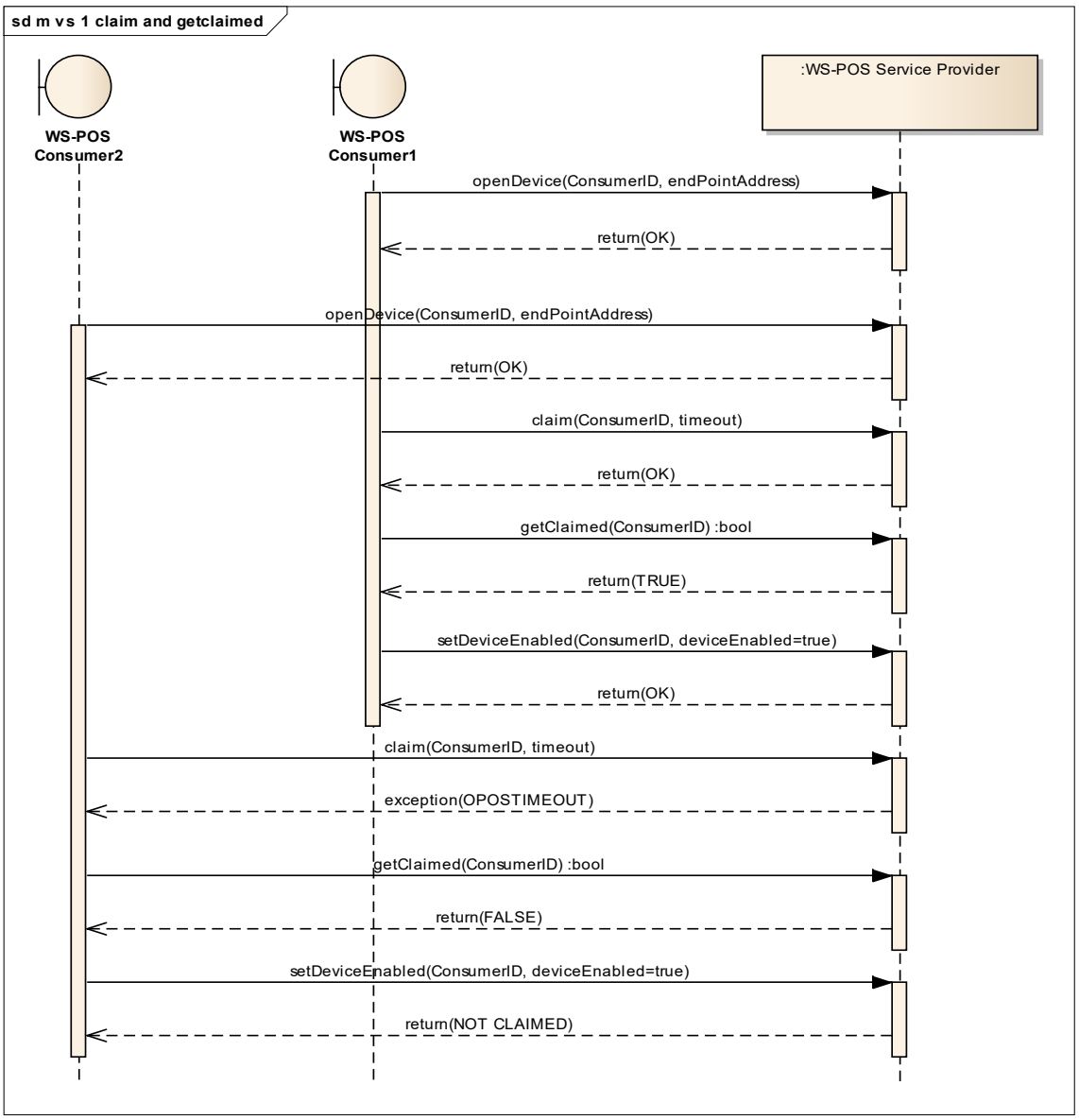


Figure 11: Claim and Get Claim

When requests have been sent to the WS-POS service provider from multiple WS-POS service consumers, the request that came from WS-POS service consumer that has executed a successful **claim** method call and optionally a device dependent **enable** method call is then able to use the functionality of the WS-POS service provider.

The following sequence diagram illustrates a scenario where WS-POS Consumer 2 cannot utilize the WS-POS service provider until WS-POS Consumer 1 has executed a successful **release** method call.

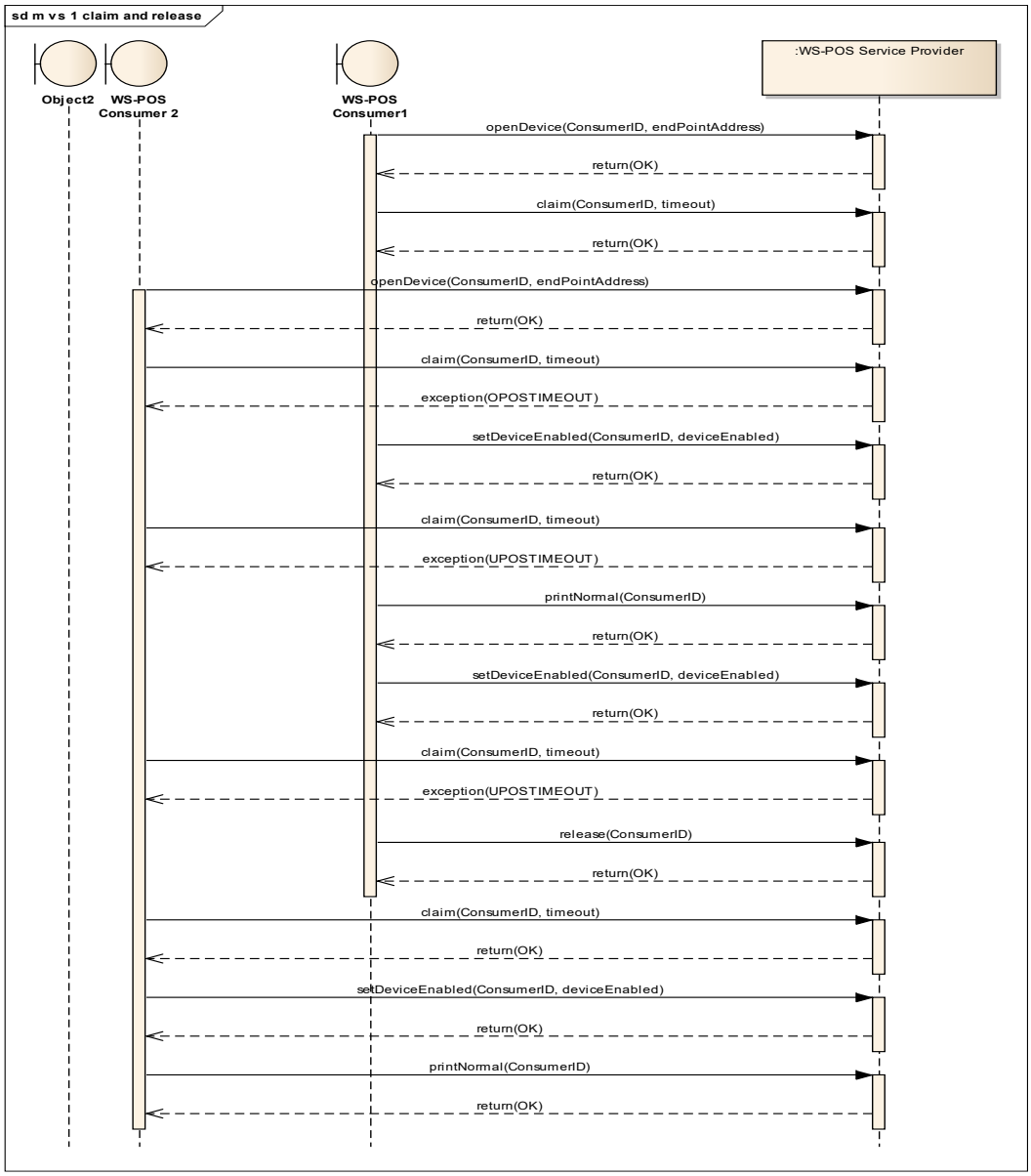


Figure 12: Claim and Release

In a complex enterprise system there may be any number of WS-POS service consumers that wish to use the functionality of a WS-POS service provider. When this is the case, an arbitration mechanism must be available in order to ensure that a WS-POS service consumer has timely access to its desired WS-POS service provider. WS-POS defines the following multiple WS-POS service consumer claim request behavior that is used to resolve WS-POS service provider contention issues. The following discussion and diagram illustrate how to handle multiple WS-POS service consumer claim requests.

1. WS-POS service consumer 1 issues a **claim** method call to the WS-POS service provider. Since no other WS-POS service consumer is currently using the requested WS-POS service provider, the **claim** method is acknowledged with a successful
   claim result.

2. Next a WS-POS service consumer 2 issues a **claim** method call to the same WS-POS service provider that WS-POS service consumer1 is currently using. The WS-POS service provider queues the request but, using a long poll methodology, does not issue any acknowledgement at this point. It uses the "Timer Value" that WS-POS service consumer 2 sent with its **claim()** method call to set a timer which will expire if WS-POS consumer1 does not finish using the WS-POS service provider before the timer expires.

3. If the timer expires, the WS-POS service provider returns a "Timer Value Expired" error code back to WS-POS service consumer2 implying the WS-POS service provider is not available for use at this time. The WS-POS service consumer2 can either issue another **claim** method call with a new "Timer Value" to the WS-POS service provider or it may decide to look for another WS-POS service provider in the enterprise that it can use for the same functionality and issue a **claim** method call to it.

4. While WS-POS service consumer1 is using the WS-POS service provider and WS-POS service consumer2 is waiting for the WS-POS service provider to become available, WS-POS service consumer3 issues a **claim** method call to the same WS-POS service provider. Now the WS-POS service provider queues up the WS-POS service consumer3 request behind WS-POS service consumer2 request and sets a timer for WS-POS service consumer3 claim request using the WS-POS service consumer3 **claim** method call "Timer Value".

5. The WS-POS service provider uses a FIFO mechanism to manage the WS-POS service consumer **claim** method call queue. If WS-POS service consumer1 is finished with the WS-POS service provider, it issues a **release** method call. Then the WS-POS service provider responds back to WS-POS service consumer2 with a successful claimed notification. WS-POS service consumer3 now moves up in the queue and waits for either a successful claimed acknowledgement or "Timer Value Expired" notification.

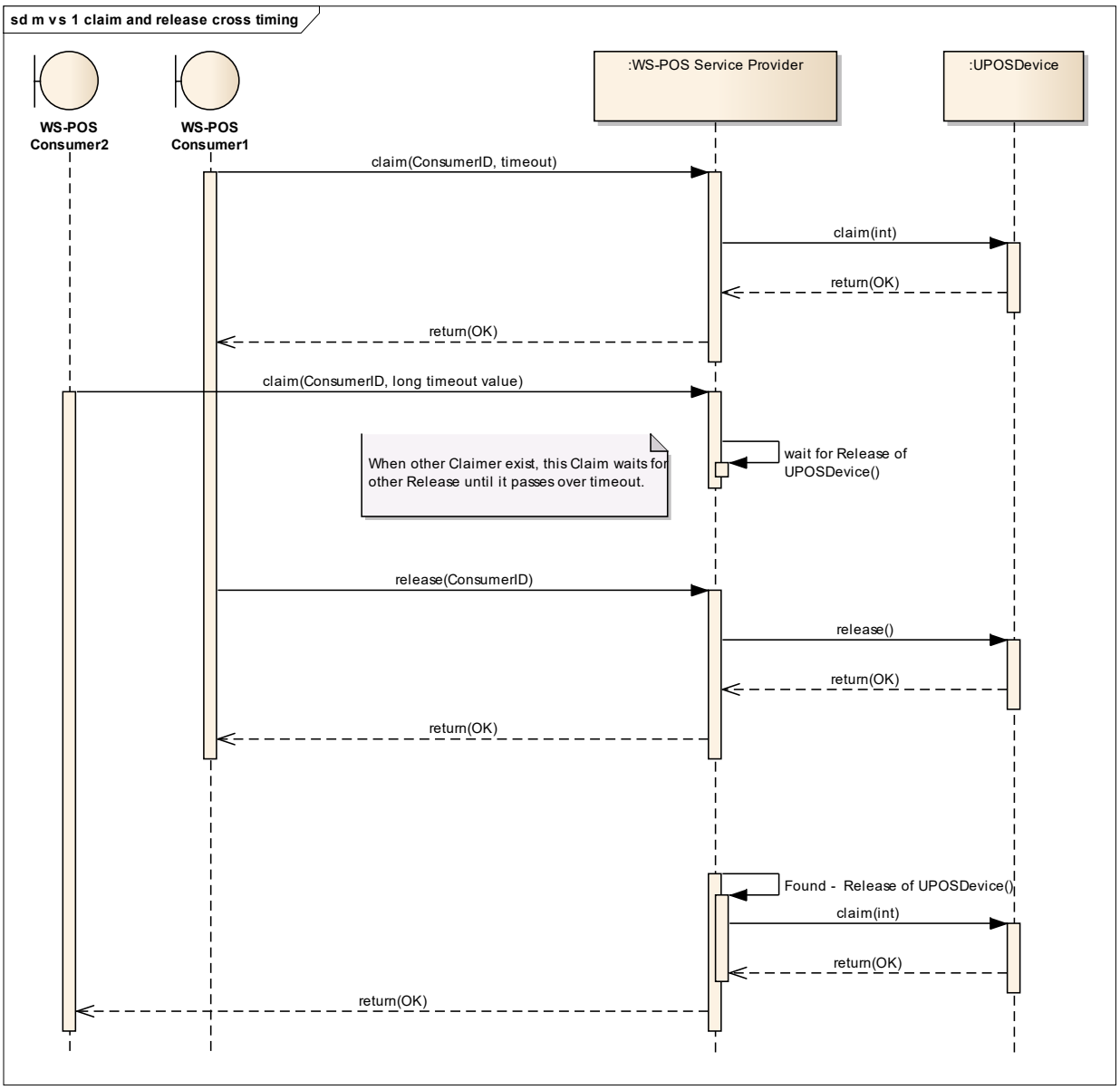


Figure 13: Claim and Release Cross Timing

2.4.9 WS-POS Methods and Device Methods *Updated in Version 1.3*

To facilitate the process of session management, WS-POS separates method calls into two groups. The "WS-POS Methods" group references accessing the Session Manager layer while the "Device Methods" group references the layer that controls the UnifiedPOS devices. The following tables summarize the applicable methods.

2.4.9.1 WS-POS Methods

Method	Description	May Use After
generateConsumerID	WS-POS service provider generates consumer ID as a GUID or a UUID, and WS-POS service consumer retrieves it.	
openSession	Starts WS-POS Session (Device is not opened)	
closeSession	Closes WS-POS Session	openSession
getProviderSessionTimeout	WS-POS service consumer retrieves a session time-out value set to WS-POS service provider.	openSession
keepAlive	WS-POS service consumer calls this method of the WS-POS service provider to maintain a session periodically.	openSession
pollForUPOSEvent	WS-POS service consumer performs polling whether or not an event occurs for WS-POS service provider.	openSession
setEventResponse	WS-POS service consumer sets a reply for the event acquired in pollForUPOSEvent.	pollForUPOSEvent
getWSPOSVersion	WS-POS consumer retrieves a version of WS-POS which WS-POS service provider implements.	
getEncryptedClaimedConsumerID	WS-POS service consumer retrieves an encrypted value of the consumerID of the session that has been successfully claimed by WS-POS service provider.	Version 1.3
setEventRequestProperties	Sets property which is notified by DataContainedEvent to WS-POS service provider. Switches DataEvent and DataContainedEvent.	Version 1.3

2.4.9.2 Device Methods

Method	Description	May Use After
openDevice	Opens UnifiedPOS Device	openSession
claim	Claims a UnifiedPOS Device	openDevice
release	Releases a UnifiedPOS Device	openDevice
closeDevice	Closes UnifiedPOS Device	openDevice
getBinaryConversion	Get the BinaryConversion property of a UnifiedPOS Device.	openDevice Version 1.3
setBinaryConversion	Set the BinaryConversion property of a UnifiedPOS Device.	openDevice Version 1.3
UnifiedPOS methods (except "open" and "close")	Access to UnifiedPOS device *(See UnifiedPOS Specification for other methods)*	openDevice (WS-POS)

2.4.9.3 Methods Not Used in WS-POS 1.2

In order to facilitate WS-POS session management, the following UnifiedPOS methods are not used in WS-POS 1.2

Method	Description
open	Instantiates a UnifiedPOS device
close	Closes an instance of a UnifiedPOS device

The function of the WS-POS "**openDevice**" starts a manager session which then creates a link to a UnifiedPOS device service object at the Device Service Layer. Similarly, the WS-POS "**closeDevice**" disconnects a manager session which removes the link to a UnifiedPOS device service object at the Device Service Layer. Therefore, in WS-POS 1.2, there is no need for the UnifiedPOS **open** and **close** methods since the **openDevice** and **closeDevice** methods accomplish the service object linking and delinking functions.

2.4.10 WS-POS Events Handling Using Bi-Directional Communication

Updated in Version 1.2

Prior to WS-POS 1.2, Event handling relied upon bi-directional communication of messages for status notification between a POS application and a POS peripheral.

WS-POS service providers may notify WS-POS service consumers of POS peripheral events defined in the UnifiedPOS standard. WS-POS service consumers who would like to receive the events for this notification must notify the WS-POS service provider of the end point address for event reception.
The process consists of:

- The WS-POS service provider notifies the Unified POS device event to the end point address for event reception registered by the WS-POS service consumer.
- The WS-POS service consumer notifies the WS-POS service provider to delete the event reception end point address when event reception is no longer necessary.

A sequence diagram of the procedure is shown in Figure 14.

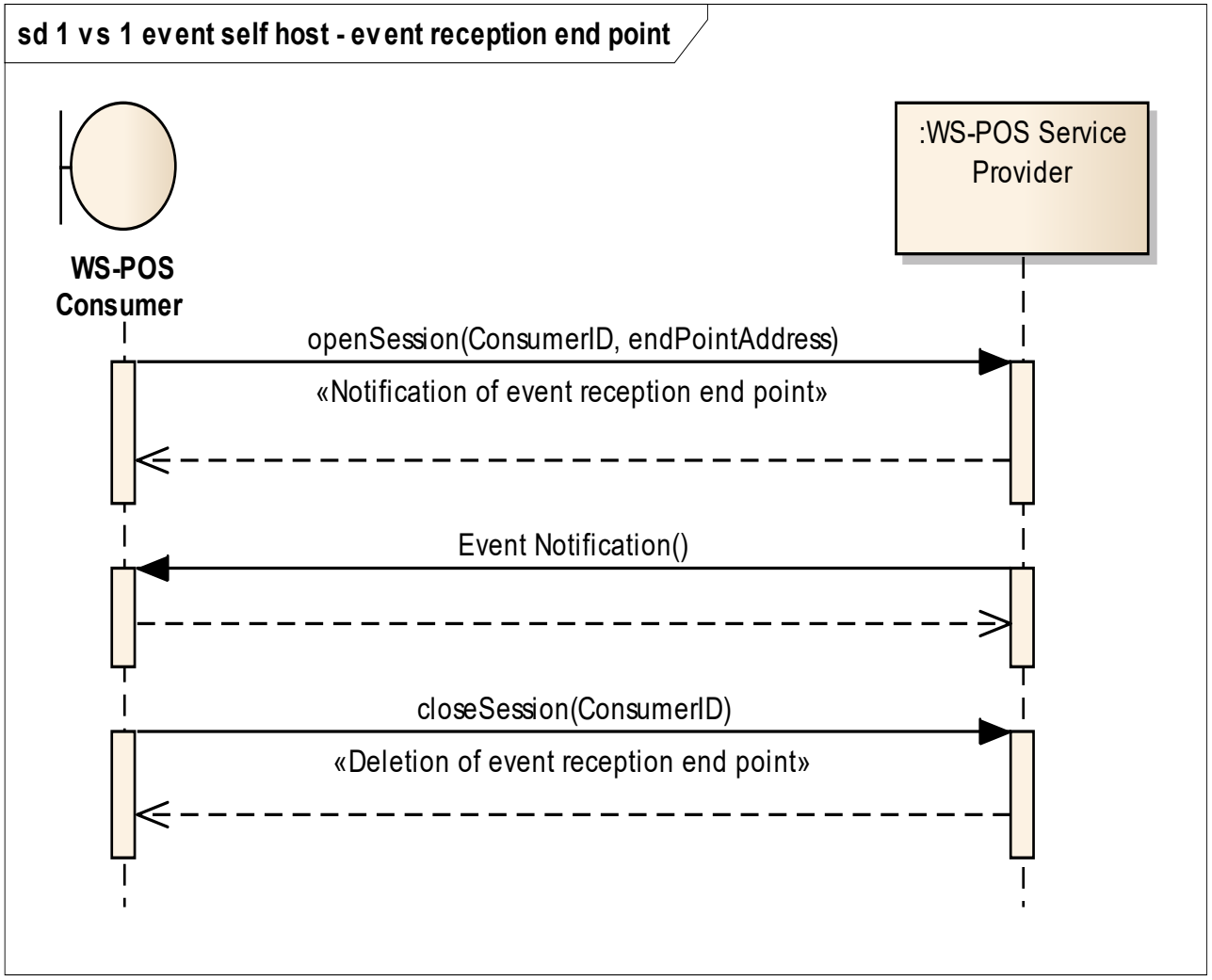


Figure 14: Event Self Host - Event Reception End Point

In WS-POS 1.2, an additional mechanism has been provided; the process of using a method call and event notification as shown below.

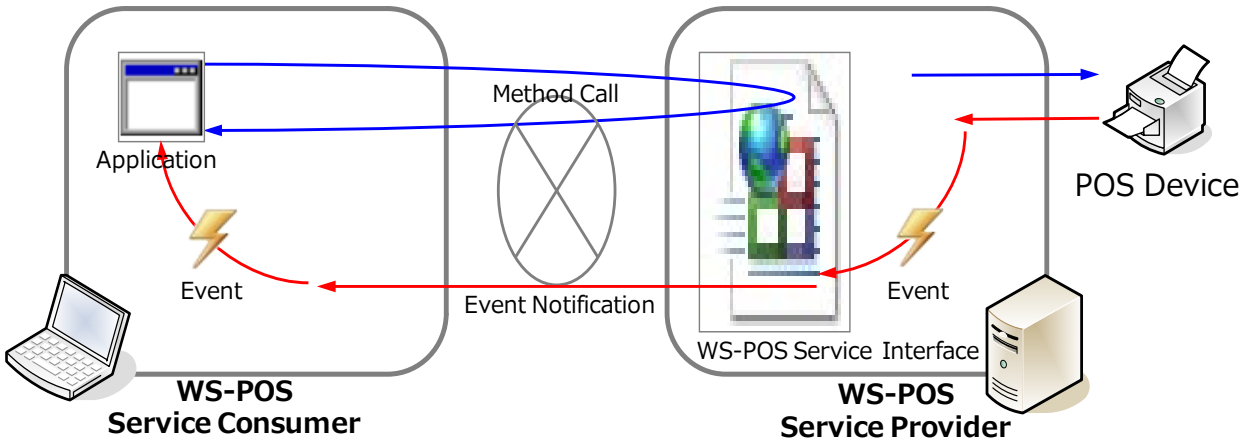


Figure 15: Method Call with Event Notification

Here the WSDL and XSD schema are defined to facilitate the WS-POS service provider sending an event notification to the WS-POS service consumer. This process requires the WS-POS service consumer to register its event end point notification address with the WS-POS service provider. When the WS-POS service provider receives a POS Device event, it sends an event notification to the WS-POS Service Consumer. Proxy code and a configuration file are used to support the specific Java or C# programming environment.

The WS-POS service consumer passes its event notification end point address to the WS-POS service provider as a parameter of openSession method. This end point address must be the entry point for the service consumer's WebService which is defined in the WSDL and XSD schema. A contract written in Java or C# is provided in the WS-POS 1.2 Support Files for simpler implementation and more support for the chosen programming environment. A contract to receive the events uses the naming convention "device class + Event". WebService methods to handle the events are described in section 2.4.17.

When the WS-POS service consumer calls the **closeSession** method, the WS-POS service provider deletes the previously registered WS-POS service consumer's event end point address.

Figure 16 describes the process for servicing a POSPrinter **ErrorEvent**.

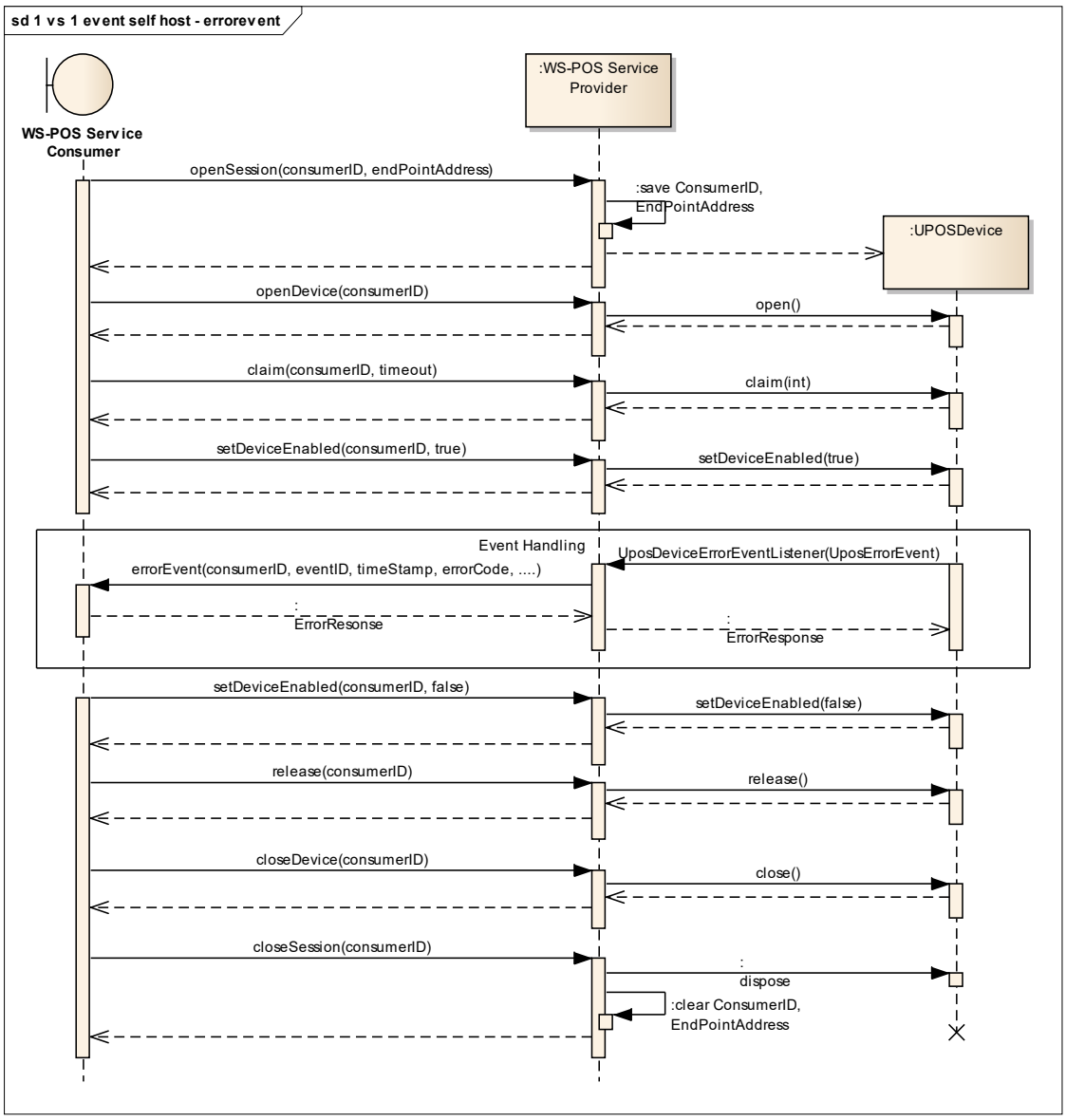


Figure 16: Event Self Host - Error Event

2.4.11 WS-POS vents Handling on Polling *Added in Version 1.2*

In WS-POS 1.2, “Polling Methodology” was added as an alternative way for the application and POS peripheral to pass status information.

In a Web Application, for reasons of browser security, there may be cases where the application cannot open a port and bi-directional communication becomes technically impossible. In these scenarios, a service consumer, running as a local application located in a local PC, may have its input and output port access blocked by its system fire wall. Notification of events between a service provider and a service consumer would never occur.

WS-POS Ver.1.2 introduces “polling” as an alternative in order to facilitate event processing. Polling is conducted by the WS-POS service consumer who wants to receive (acquire) the POS peripheral device events as defined by the UnifiedPOS standard. A polling sequence is used to see whether or not the event for which the WS-POS service provider should be making notification is being requested.

In order to assure timely event processing using this polling technique, the polling time interval (frequency of polling) is adjustable. The required polling time interval must be adjusted in accordance with the goals of an optimum system configuration.

When the poll by the WS-POS service consumer determines that the WS-POS service provider has queued an event, the WS-POS service consumer processes the poll event using similar response processing it would have used for a WS-POS service provider interrupt type event.

Once the event notification has been received, the WS-POS service consumer formulates the appropriate event response to send to the WS-POS service provider. This event response may include any of the UnifiedPOS defined actions for **ErrorEvent**s and **StatusUpdateEvent**s.

The WS-POS service provider may have any number of queued events. The WS-POS service consumer should process the poll events systematically and quickly. The WS-POS service provider, having received an event response from the WS-POS service consumer, and depending on the type of event that has been generated, may immediately notify the WS-POS service consumer of another event.

The WS-POS service consumer may terminate the polling when event reception (acquisition) is no longer required.

Figure 17 illustrates a typical event processing using an **ErrorEvent** as an example.

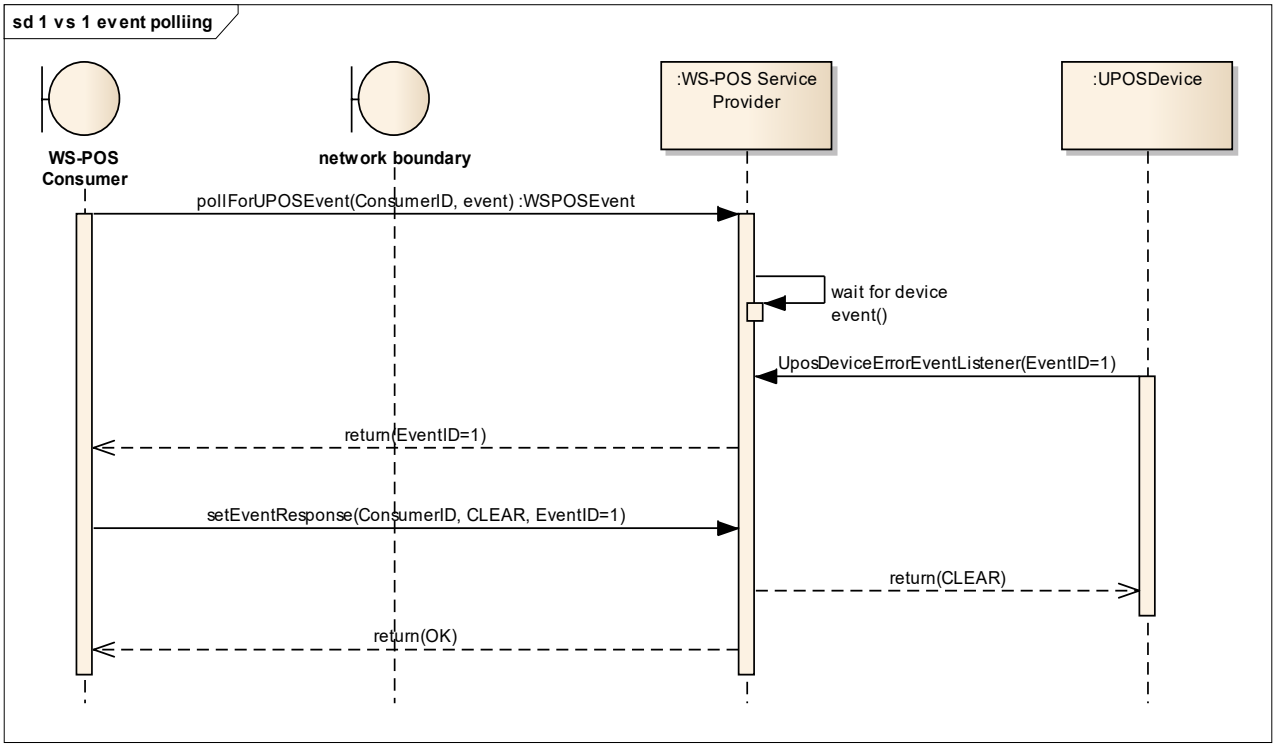


Figure 17: Event Polling

Note that the WS-POS service consumer notifies the WS-POS service provider to clear the error after the WS-POS service consumer has received the **ErrorEvent**.

Figure 18 shows the relationship of a method call and corresponding event notification.

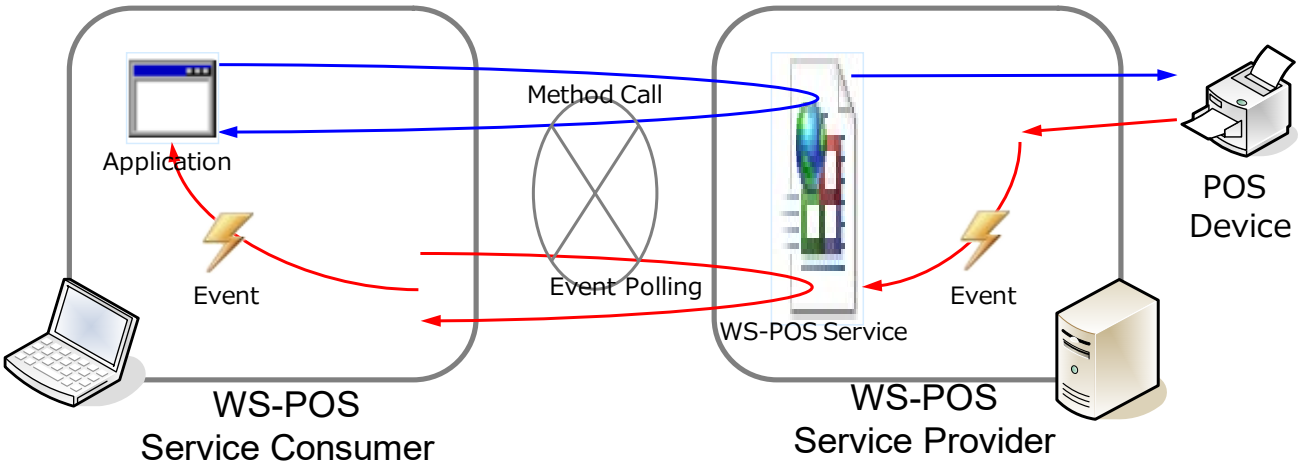


Figure 18 Method call and Event notification

In this example, the event polling by the WS-POS service consumer for the purpose of event reception and receipt by the WS-POS provider is supported by the programming environment and is realized by the proxy code and configuration file by the WS-POS Service.

The polling from the WS-POS to the service provider uses a "long polling" methodology *(WS-POS Service Provider holds poll request if event is not pending or until a suitable timeout occurs)*. This is executed to reduce the load affecting system operations that have been triggered by service consumer and service provider processing.

The standard polling concept is shown in Figure 19.

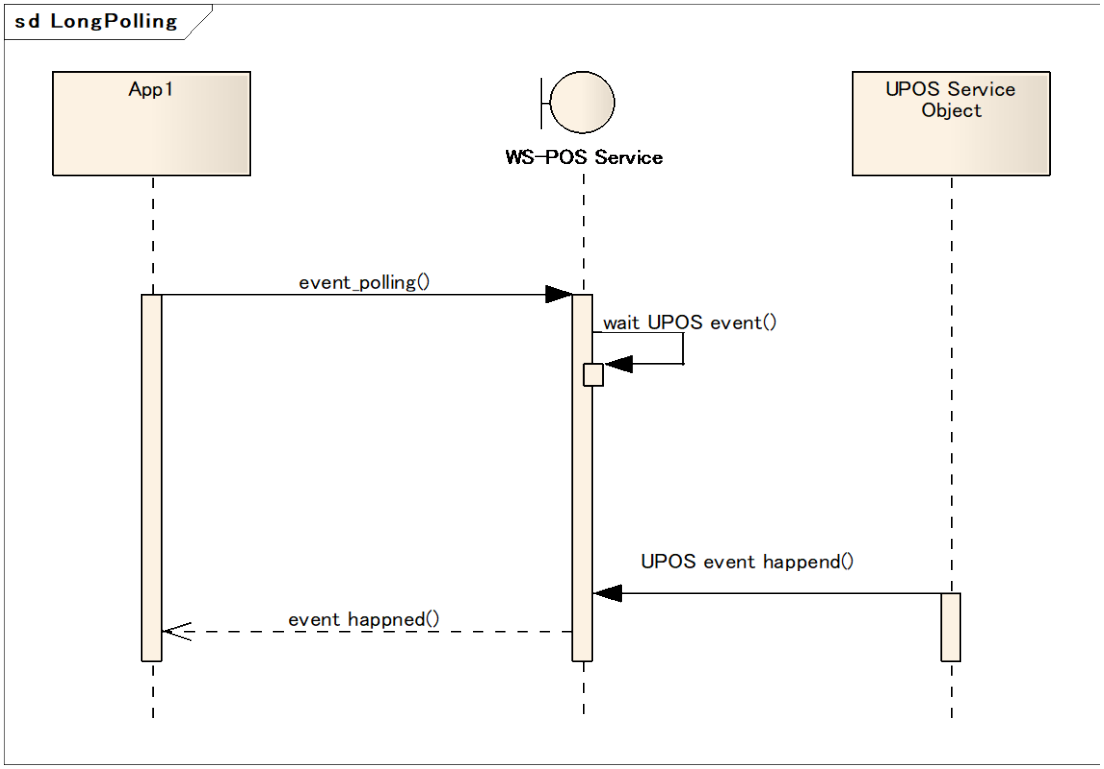


Figure 19: Long Polling Overview

When employing long polling, a service provider does not immediately return a “no event” message when no event has occurred; instead it waits for an event or a timeout to be generated before it returns a response message. In this way traffic and overhead processing loads can be reduced. Ordinary polling requires that an immediate response of “event” or “no event” be sent by the service provider to the service consumer upon receipt of a poll.

When an event has not been generated over a long period of time, the polling wait reaches time-out in the transport layer. If the polling times out, the WS-POS service consumer immediately resumes polling. This behavior is shone in Figure 20.

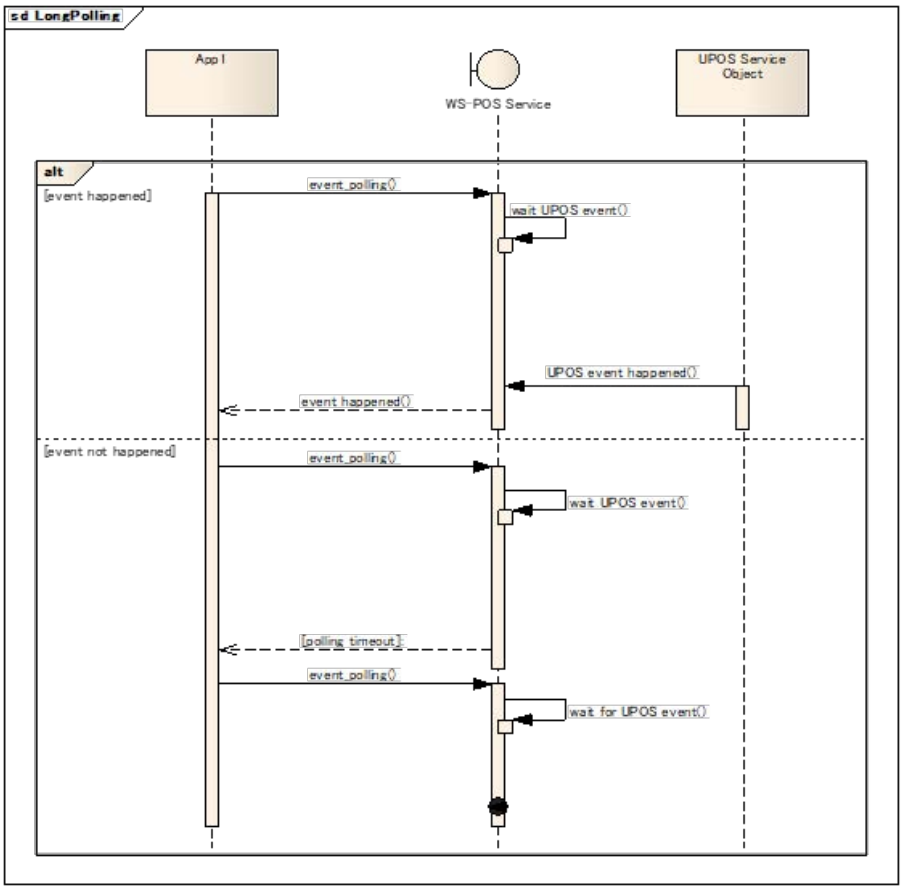


Figure 20: Long Polling

2.4.12 Resolution of frequent communication events in the Event notification

Added in Version 1.3

Up to and including WS-POS version 1.2, the event handling in a WS-POS service consumer utilized the following sequence:

1. DataEventEnabled is set to *false* at initial state. Events are enqueued.
2. When the WS-POS service consumer sets its dataEventEnabled to *true*, the WS-POS service provider sets the dataEventEnabled to *false*, prepares for enqueuing following event, then notifies the events enqueued event to the WS-POS service consumer.
3. The WS-POS service consumer acquires the property that was indicated by the event message and responds appropriately.
4. The WS-POS service consumer sets its dataEventEnabled to *true*, and is ready for next WS-POS service provider event.

This process works well for most situations. However, it is possible for some devices, a POS Keyboard for example, to generate a large number of events in a short period of time. This can cause a large backlog of the events, heavy network traffic with short "chatty" notification instances and a noticeable drop in system performance. This becomes more of a problem when the WS-POS service provider and the WS-POS service consumer are connected on different nodes on a large physical network.

In the case of POSKeyboard, a series of single keystrokes can cause a large amount of network traffic. Notice how this creates the following additional, network intensive, sub-steps.

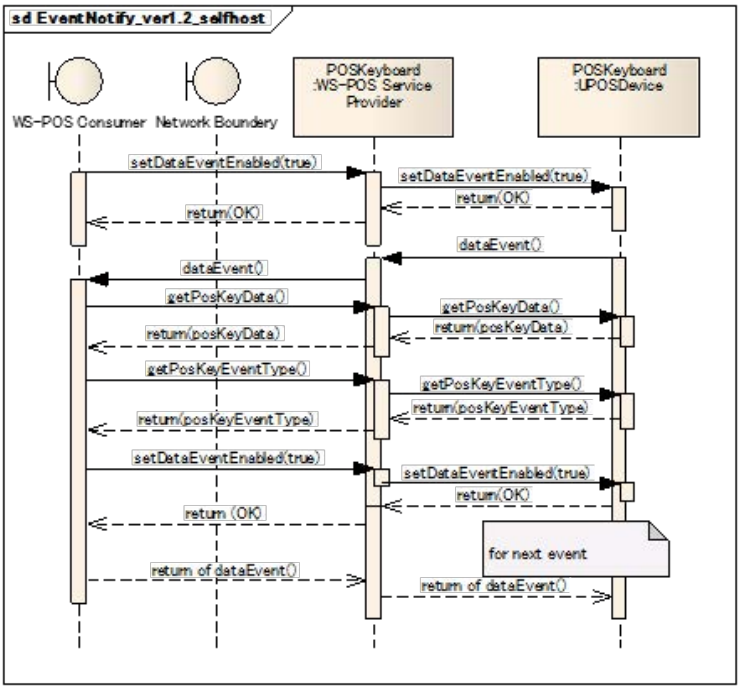
1. A single POSKeyboard key is depressed and a dataEvent message is generated by the POSKeyboard WS-POS service provider for a specific WS-POS service consumer.
2. The WS-POS service provider sets its dataEventEnabled property to *false* before notifying dataEvent, then sends the event to the WS-POS service consumer.
3. In response, a message by the WS-POS service consumer is sent to the WS-POS service provider requesting the posKeyEventType property.
4. The WS-POS service provider responds with a message back to the WS-POS service consumer with the posKeyEventType property.
5. If the posKeyEventType property indicates that it has POS keyboard "key" data, the WS-POS service consumer sends another message back to the WS-POS service provider to send the posKeyData property value.
6. The WS-POS service provider then sends back a message to the WS-POS service consumer with the posKeyData property value.
7. The WS-POS service consumer sets the dataEventEnabled to *true* and sends this in a message back to the WS-POS service provider indicating that the WS-POS service consumer is ready for the next event.
8. If another key has been depressed this sequence starts over again.

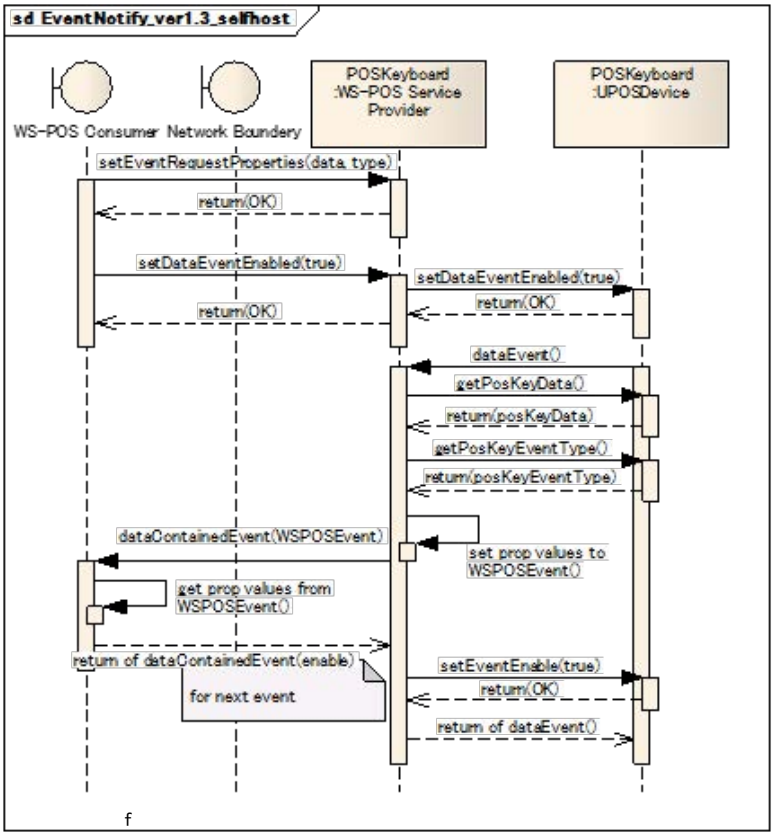
Depending upon the network connection and network traffic load factors, this "chatty" messaging system degrades overall system performance resulting in slow and unsatisfactory WS-POS based applications.

In WS-POS version 1.3, the sequence and data structure in the Event notification is improved to reduce the network traffic necessary for data transmission between the WS-POS service consumer and the WS-POS service provider. A new property value, dataContainedEvent has been introduced. Now, when the WS-POS service provider sends an event to the WS-POS service consumer, a new dataContainedEvent parameter can be used instead of the dataEvent parameter. The value of dataContainedEvent is used to include the property value that is associated with the event. The WS-POS service consumer can specify whether setDataEventEnabled property is set to true or not and clearInputProperties method is called or not by evaluating the value of dataContainedEvent property.

A WS-POS service consumer can specify the value of the property delivered by the dataContainedEvent using the setEventRequestProperties method. If nothing is specified, the default value to be transmitted is dataEvent instead dataContainedEvent.
During the polling process, the property value which is changed by dataEvent is included in WSPOSEvent, which is the return value from a pollForUPOSEvent method. Also, whether dataEventEnabled property is set to true or not and whether clearInputProperties method can be called or not, is specified by the parameter associated by the setEventResponse.

The following figure is a comparison of the dataEvent sequence prior to WS-POS version 1.3 for bi-directional communication. It shows the POSKeyboard as an example. When dataEvent is fired, posKeyData and posKeyEventType property is retrieved, then set dataEventEnabled property to true so that next dataEvent can be retrieved. This results in fewer messages and potentially better network performance.


sd EventNotify_ver1.2_selfhost
WS-POS Consumer
Network Boundery
POSKeyboard :WS-POS Service Provider
POSKeyboard :UPOSDevice
setDataEventEnabled(true)
setDataEventEnabled(true)
return(OK)
return(OK)
dataEvent()
dataEvent()
getPosKeyData()
getPosKeyData()
return(posKeyData)
return(posKeyData)
getPosKeyEventType()
getPosKeyEventType()
return(posKeyEventType)
return(posKeyEventType)
setDataEventEnabled(true)
setDataEventEnabled(true)
return(OK)
return (OK)
for next event
return of dataEvent()
return of dataEvent()


sd EventNotify_ver1.3_selfhost
WS-POS Consumer
Network Boundery
POSKeyboard :WS-POS Service Provider
POSKeyboard :UPOSDevice
setEventRequestProperties(data type)
return(OK)
setDataEventEnabled(true)
setDataEventEnabled(true)
return(OK)
return(OK)
dataEvent()
getPosKeyData()
return(posKeyData)
getPosKeyEventType()
return(posKeyEventType)
dataContainedEvent(WSPOSEvent)
set prop values to WSPOSEvent()
get prop values from WSPOSEvent()
return of dataContainedEvent(enable)
for next event
setEventEnable(true)
return(OK)
return of dataEvent()

f

The following figure is comparison of the dataEvent sequence prior to WS-POS version 1.3 for polling method. It shows the POSKeyboard as an example. When dataEvent is fired, posKeyData and posKeyEventType property is retrieved, then set dataEventEnabled property to true so that next dataEvent can be retrieved. This results in fewer messages and potentially better network performance.

2.4.13 WS-POS Service Network Connection Management Considerations

Added in Version 1.2

Situations can arise where there is a communication breakdown between a WS-POS service consumer and a WS-POS service provider. The problem could reside in the WS-POS service consumer, the WS-POS service provider, or a break in the network connection that ties them together. Detecting such error conditions, programmatically uncoupling WS-POS service consumers from the WS-POS service providers, and providing network connection resilience becomes a necessity. Note that in order to be successful, the WS-POS service provider must be able to detect the status of the WS-POS service consumer and the WS-POS service consumer must be able to detect the status of the WS-POS service provider. WS-POS 1.2 added a "**WS-POS Keep Alive**" mechanism to detect and resolve these process and communication disruptions.

2.4.13.1 WS-POS Service Provider Detection of an Interrupted Connection with a WS-POS Service Consumer

Initially, a WS-POS service consumer will request a connection to a WS-POS service provider by using the **openSession** method call. The normal result will be a response back from the WS-POS service provider that a successful connection has been made.

The WS-POS service provider can detect that the WS-POS service consumer is still active and connected if it can rely on the WS-POS service consumer to periodically issue any of the UnifiedPOS "valid after **open**" requests, including device status queries.

The WS-POS service consumer will send these requests at a programmable time interval defined as the "Keep Alive Interval".

The WS-POS service provider expects a WS-POS service consumer request within a programmable time interval defined as the "Provider Session Time-out".

If the WS-POS service provider receives a WS-POS service consumer request before a Provider Session Time-out expires, it will reset and restart its internal timer, process the command, send an event notification back, and wait for a new WS-POS service consumer request. The WS-POS service consumer will also reset its timer to the Keep Alive Interval value. Note that the Provider Session Time-out timer value must be greater than the Keep Alive Interval timer value.

If the WS-POS service provider does not receive such a request within the Provider Session Time-out, the WS-POS service provider will assume the connection has been lost and will respond as though it had received a **closeSession** method request from the WS-POS service consumer.

The Provider Session Time-out value is determined and set externally at Application Install and System Configuration time. For example, in a .NET Framework/WCF implementation, the Provider Session Timeout value is loaded into the .config file. In a JAX-WS implementation, the Provider Session Time-out value is set in the properties file under WEB-INF\classes.

The WS-POS service consumer inputs the Provider Session Time-out value when it makes reference to the **ProviderSessionTimeout** parameter. It then uses this value to calculate an appropriate timer value for the **KeepAliveInterval** parameter.

There may be multiple WS-POS service consumers communicating with a single WS-POS service provider when a communication breakdown occurs. The WS-POS service provider must institute error recover processes for each of these WS-POS service consumers connected to it at the time of the communication breakdown.

Figure 21 illustrates how the WS-POS service provider would respond when a communication breakdown occurs and error recovery is required.

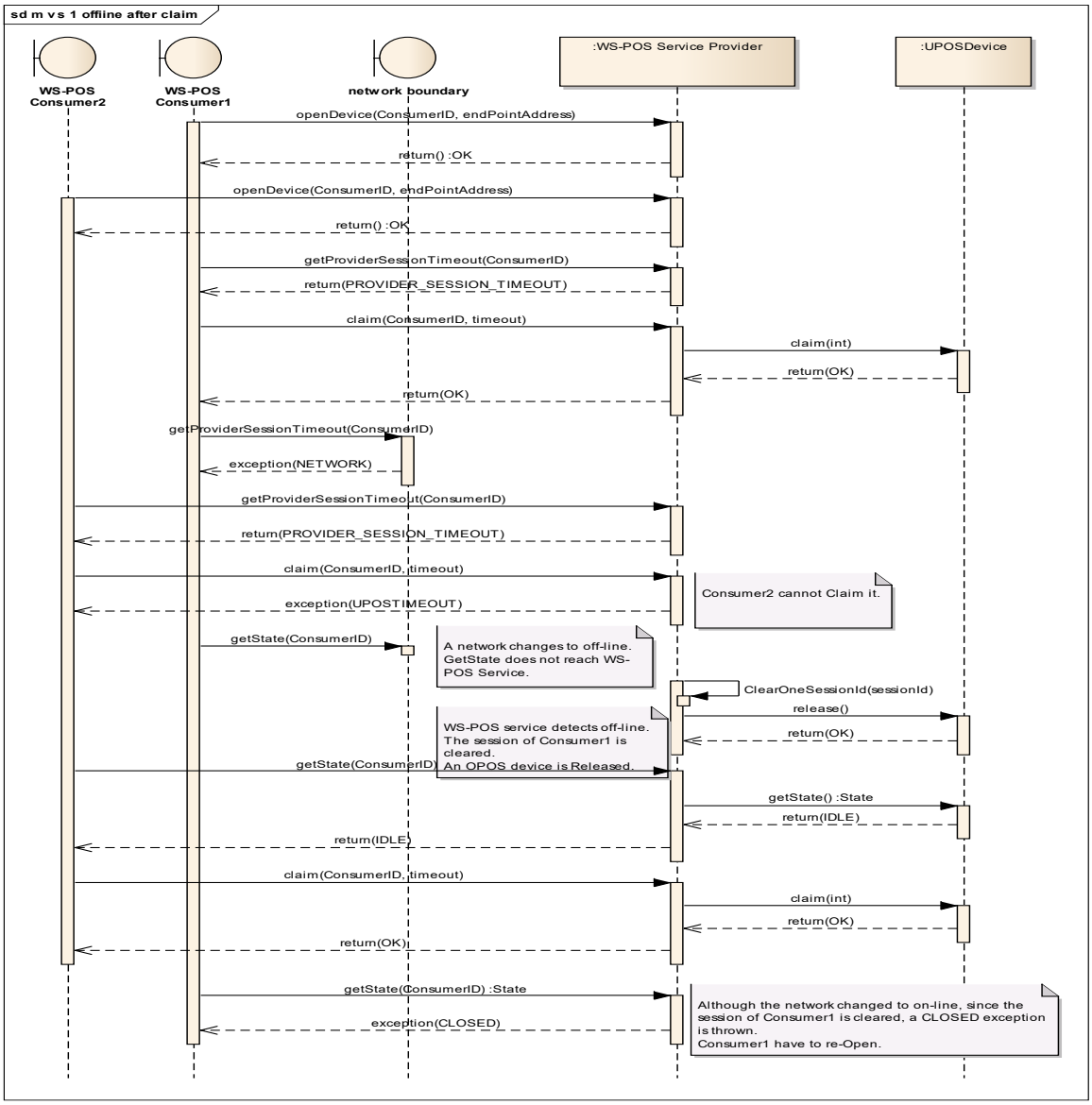


Figure 21: Offline after Claim

In the case where the WS-POS service provider detects that the network communication is disconnected, it will automatically respond as though it has received a **closeSession** method call from each of its WS-POS service consumers. After network communication is restored, the WS-POS service consumers are required to issue **openSession** method calls to the WS-POS service provider(s) they were using before the network interruption occurred.

2.4.13.2 WS-POS Service Consumer Detection of an Interrupted Connection with a WS-POS Service Provider

Initially, a WS-POS service consumer will issue an **openSession** method call to a WS-POS service provider it wishes to use. The normal result will be a response back from the WS-POS service provider that a successful connection has been made. The WS-POS service consumer then uses the WS-POS Keep Alive mechanism to allow the WS-POS service provider that its connection is active.

But the WS-POS service consumer must also be able to determine that an active connection with the WS-POS service provider still exists. This process is handled as follows:

- The WS-POS service consumer sends a request to the WS-POS service provider, sets a timer to the Keep Alive Interval value, and waits for the corresponding event notification back from the WS-POS service provider.
- In the case where a network communication breakdown occurs, the expected event is never received by the WS-POS service consumer. Instead the WS-POS service consumer Keep Alive Interval timer expires and generates an error event.
- The WS-POS service consumer understands that this error event means the communication with the WS-POS service provider has been interrupted. It reverts back to a state before a **openSession** method call was sent to the WS-POS service provider.
- The WS-POS service consumer may send a new **openSession** method call request in order to connect to the WS-POS service provider or look elsewhere for a different WS-POS service provider to provide the required services.

2.4.14 WS-POS Service Network Connection Management, Event – Bi-directional Communications *Added in Version 1.2*

Network communication disconnect conditions can result in a number of bi-directional event error scenarios. The following examples illustrate how recovery from an **ErrorEvent** would be handled in each case.

2.4.14.1 Service Provider and Consumer Disconnection and Session Does Not Time Out

Scenario: The event notification from the WS-POS service provider to the WS-POS service consumer is not received. The Network communication is recovered prior to the WS-POS service provider session time-out value.

In Figure 22, when an **ErrorEvent** is generated and the network is off-line, the WS-POS service provider will re-try notification. An identical EventID is assigned and returned.

Note that the network communication is recovered prior to detecting a disconnection which would result in a session time-out event.

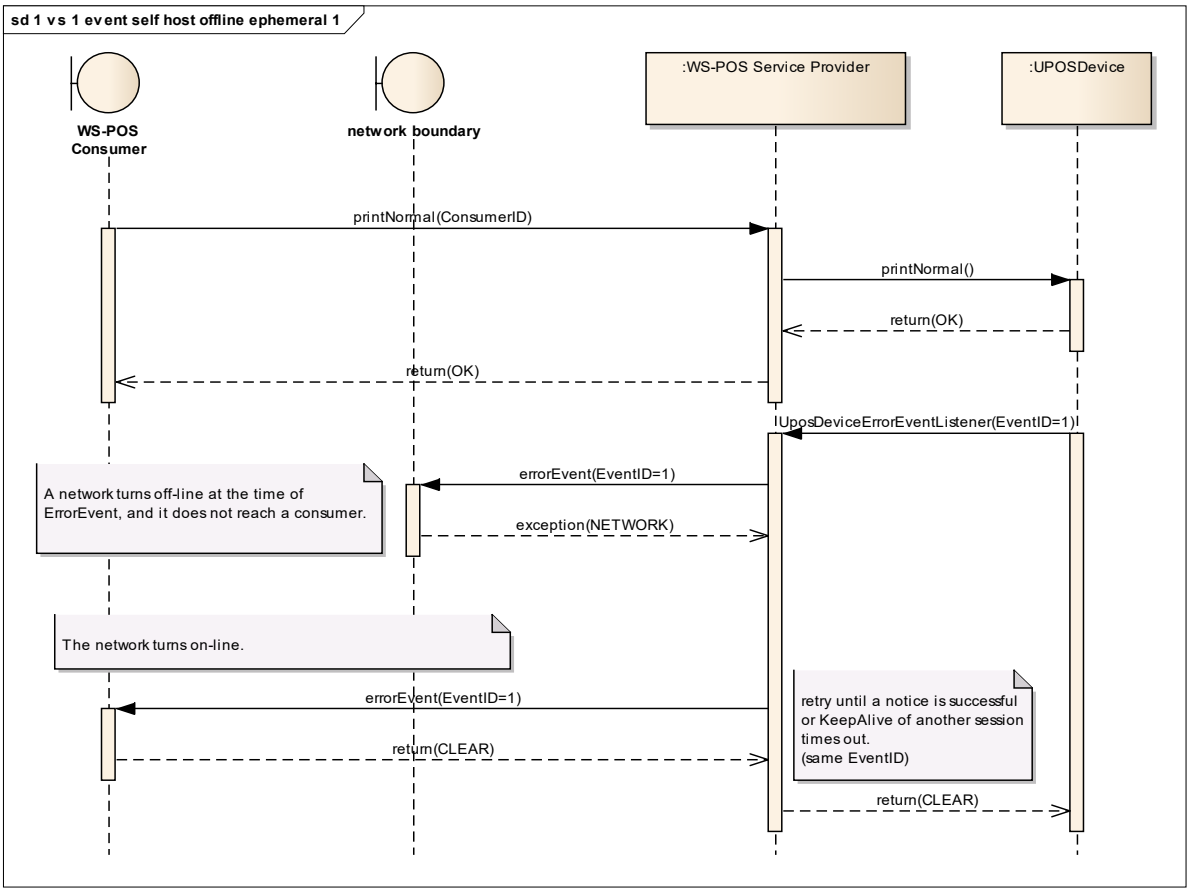


Figure 22: Offline Shorter than Session Time Out in Bi-Directional Communication 1

2.4.14.2 Consumer and Service Provider Disconnection and Session Does Not Time Out

Scenario: The event notification from the WS-POS service provider to the WS-POS service consumer is received but the response from the WS-POS service consumer does not reach the WS-POS service provider. Network communication is recovered prior to reaching the WS-POS service provider session time-out value.

When an **ErrorEvent** is generated and the network is off-line, the WS-POS service provider will re-try notification. At this time an EventID is assigned, as shown in Figure 23.

Note that the WS-POS service provider re-tries because it cannot determine if the network went off-line when sending an event or when receiving a response. This may result in the WS-POS service consumer sending multiple event notifications for events with the same EventID.

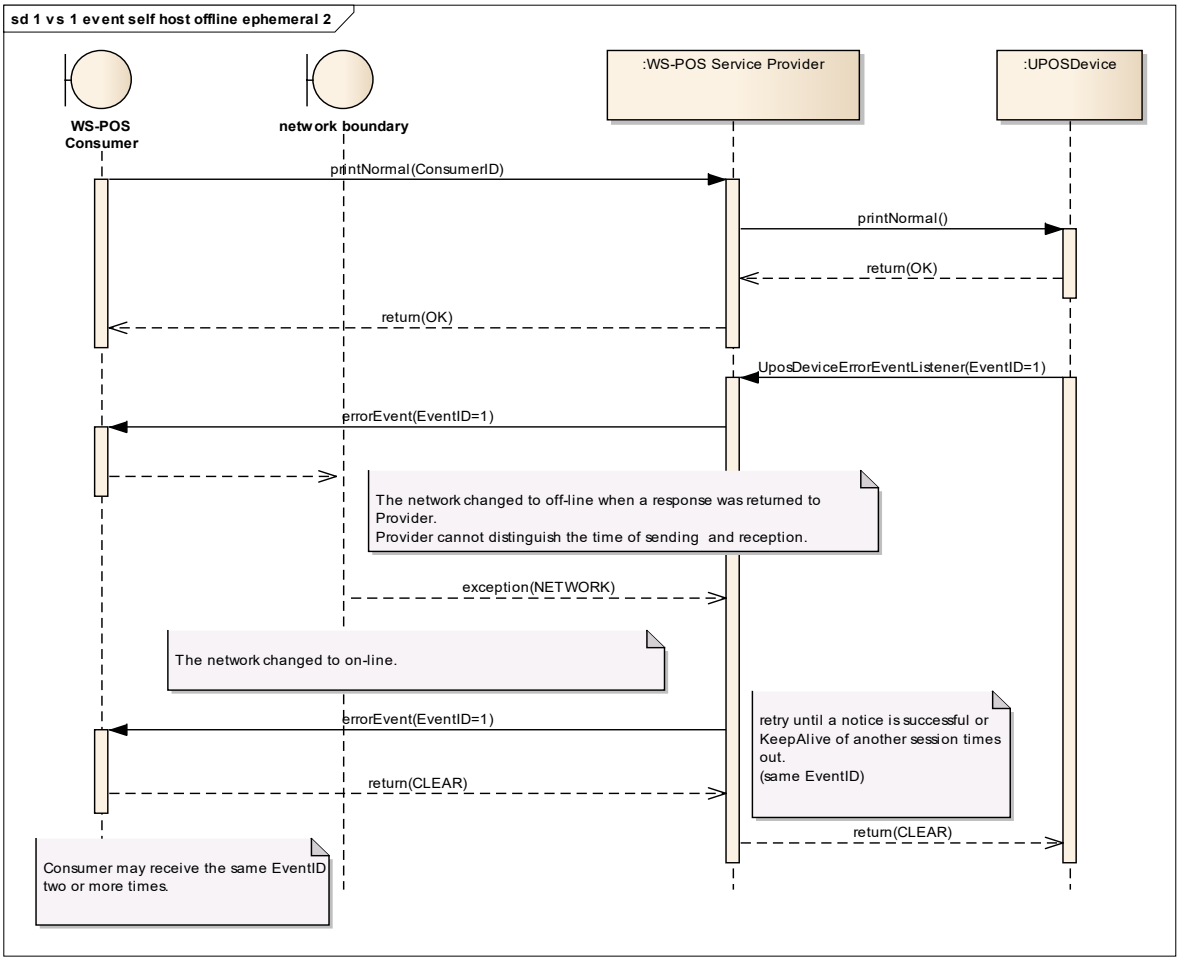


Figure 23: Offline Shorter than Session Time Out in Bi-Directional Communication 2

2.4.14.3 Service Provider and Consumer Disconnection and Session Time Out

Scenario: The event notification from the WS-POS service provider to the WS-POS service consumer is not received. Network communication is not recovered even if the WS-POS service provider session time-out value is exceeded.

In case the WS-POS provider session time out value is exceeded and Network communication is not recovered, the WS-POS service provider terminates the session and responds to event notifications from UnifiedPOS devices. For **ErrorEvents**, a **clear** is assigned in the response to the UnifiedPOS device.

Figure 24 illustrates this scenario. Note that in this case, the WS-POS service consumer is unable to receive an **ErrorEvent** even if Network communication is recovered. The WS-POS service provider has cleared the session and the WS-POS service consumer must re-execute the **openSession** method call.

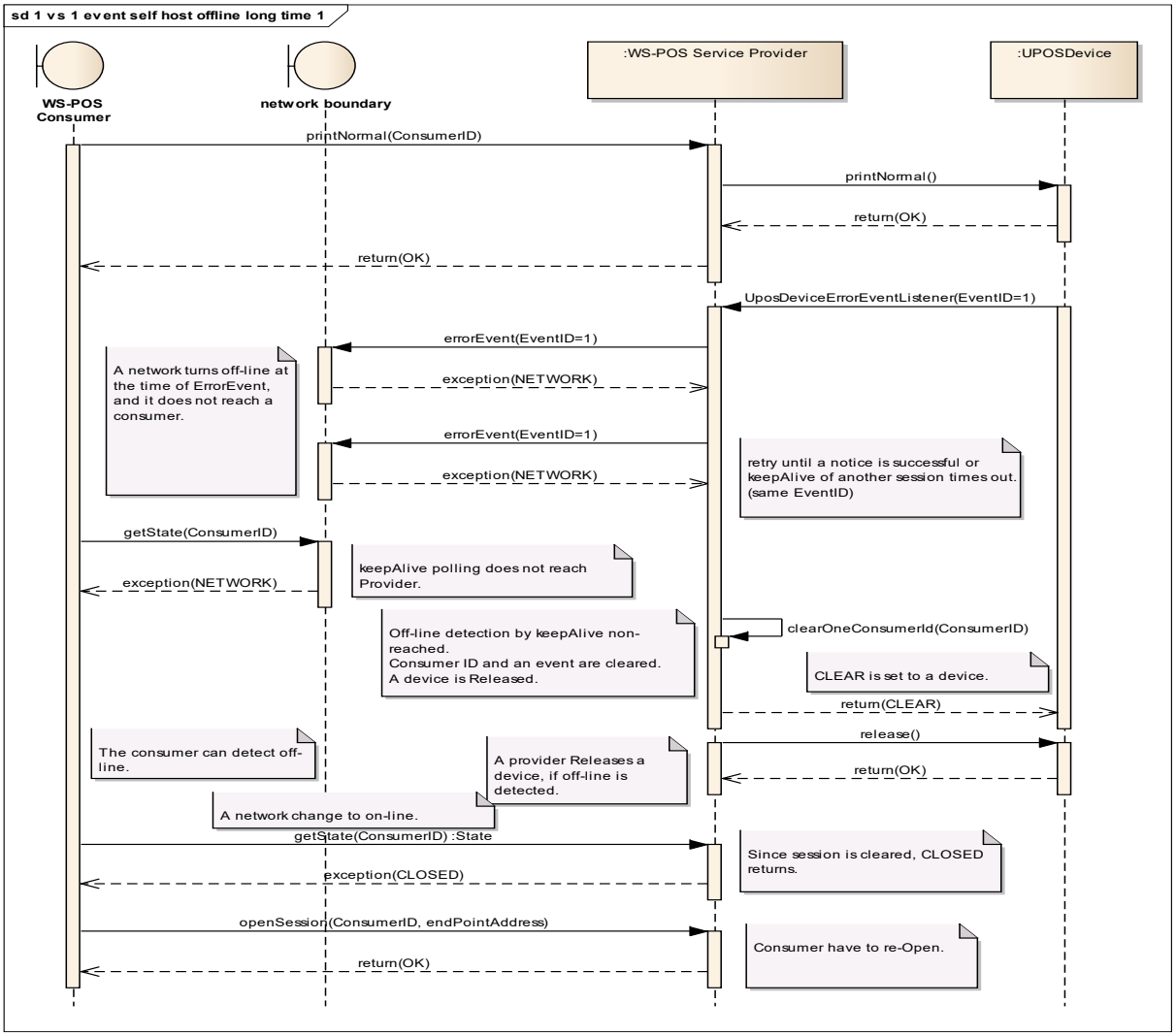


Figure 24: Offline Longer than Session Time Out in Bi-Directional Communication 1

2.4.14.4 Consumer and Service Provider Disconnection and Session Time Out

Scenario: The event notification from the WS-POS service provider to the WS-POS service consumer is received but the response from the WS-POS service consumer does not reach the WS-POS service provider. Network communication is not recovered even if the WS-POS service provider session time-out value is exceeded.

In the event that the WS-POS provider session time-out value is exceeded and Network communication is not recovered, the WS-POS service provider terminates the session and responds to event notifications from UnifiedPOS devices. In case of **ErrorEvents**, a **clear** is assigned in the response to the UnifiedPOS device.

Note that in this case, the WS-POS service consumer receives an **ErrorEvent** but, unable to get a response to the WS-POS service provider, the WS-POS service provider assigns a **clear** in the response to the UnifiedPOS device.

Even if Network communication is recovered, the WS-POS service provider is clearing the session as though a **closeSession** method call was received so the WS-POS service consumer must re-execute an **openSession** method call to reestablish a connection.

Figure 25 illustrates this scenario.

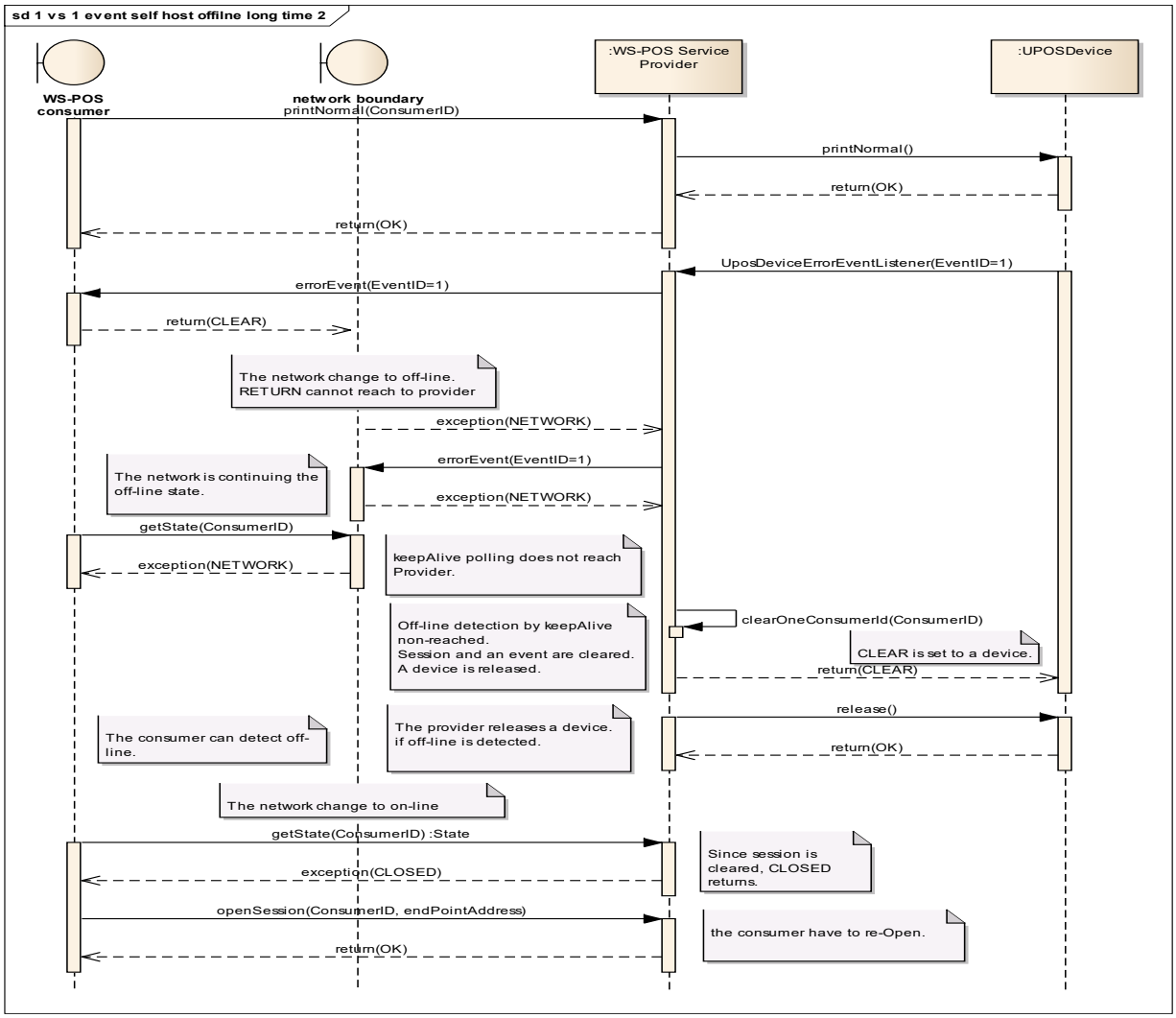
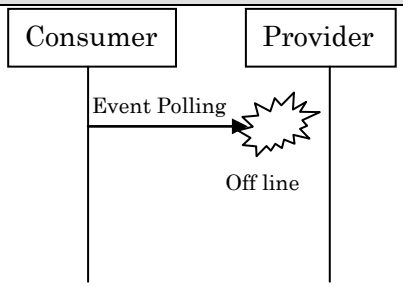
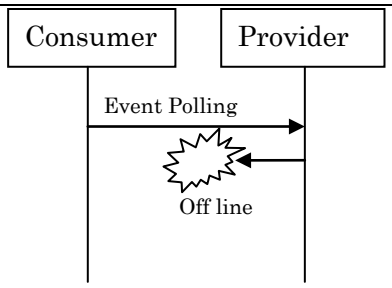
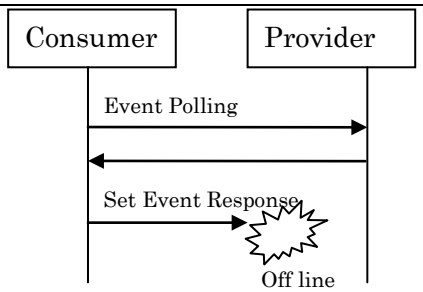


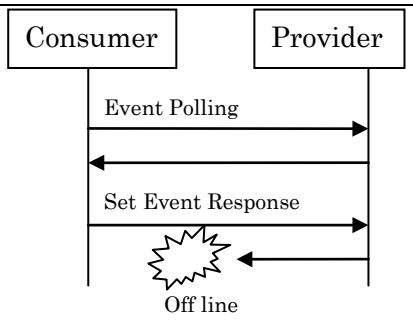
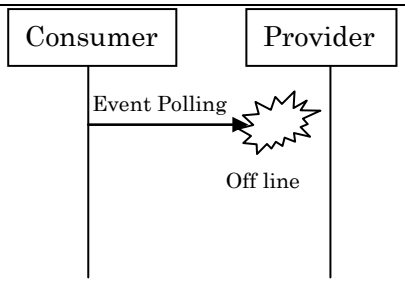
Figure 25: Offline Longer than Session Time Out in Bi-Directional Communication 2

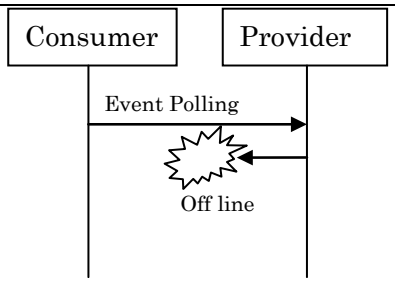
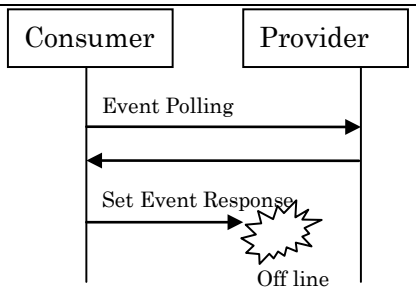
2.4.15 WS-POS Service Network Connection Management, Event – Polling

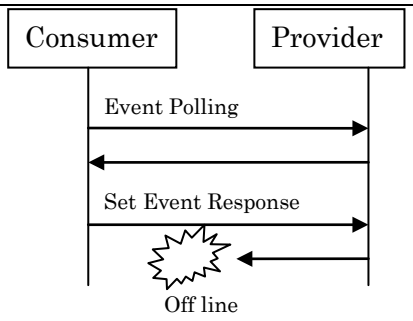
Added in Version 1.2

The network connection management must be consistent between the WS-POS service consumer and the WS-POS service provider. The conditions under which network communication disconnects during event notification, using polling, are classified and their behaviors described in the chart below:

	Explanation	Sequence Diagram	Behavior
A)	Event polling from the WS-POS service consumer does not reach the WS-POS service provider. Network communication is recovered prior to reaching the WS-POS service provider session time-out value.		The WS-POS service consumer re-executes event polling. If network communication is recovered, normal transmission conditions will be resumed with no error recovery necessary.
B)	Event polling from the WS-POS consumer reaches the WS-POS service provider but the event polling return value from the WS-POS service provider does not reach the WS-POS service consumer. Network communication is recovered prior to reaching the WS-POS provider session time-out value.		The WS-POS service consumer re-executes event polling. If network communication is recovered, normal transmission conditions will be resumed with no error recovery necessary. Refer to Section 2.4.14.1: Event polling return value is not received and session does not time out.
C)	The polling of the WS-POS service provider from the WS-POS service consumer concludes normally, but the **setEventResponse** is not received. Network communication is recovered prior to reaching the WS-POS provider session time-out value.		The WS-POS service consumer re-executes the **setEventResponse**. If Network communication is recovered, normal transmission conditions will be resumed with no error recovery necessary.

D)	The polling of the WS-POS service provider from the WS-POS service consumer concludes normally and the event response setting also is received, but the return for the **setEventResponse** does not reach the WS-POS service consumer. Network communication is recovered prior to reaching the WS-POS provider session time-out value.		The WS-POS service consumer re-executes the **setEventResponse**. In case Network communication is recovered, it sends back an ILLEGAL notification as the **setEventResponse**. Thereafter it will be the same as a normal sequence. Refer to Section 2.4.14.2: The return value for the **setEventResponse** is not received and the session does not time out.
E)	The event polling from the WS-POS service consumer does not reach the WS-POS service provider. Network communication is not recovered and the WS-POS provider session time-out value is exceeded.		The WS-POS service consumer re-executes event polling but network communication is not recovered even if the provider session time-out value is exceeded. The WS-POS service consumer determines that the session with the WS-POS service provider has been lost. The WS-POS service provider clears the session and releases the UnifiedPOS devices. When the network communication has recovered, the WS-POS service consumer reissues an **openSession** method call. Thereafter it will be the same as a normal sequence.

F)	Event polling from the WS-POS consumer reaches the WS-POS service provider but the event polling return value does not reach the WS-POS service consumer. Network communication is not recovered and the WS-POS provider session time-out value is exceeded.		The WS-POS service consumer re-executes event polling but network communication is not recovered even if the provider session time-out value is exceeded. The WS-POS service consumer determines that the session with the WS-POS service provider has been lost. The WS-POS service provider clears the session and releases the UnifiedPOS devices. When the network communication has recovered, the WS-POS service consumer reissues an **openSession** method call. Thereafter it will be the same as a normal sequence. Refer to Section 2.4.14.3: Event polling is not received and the session times out.
G)	The polling of the WS-POS service provider from the WS-POS service consumer concludes normally but the **setEventResponse** is not received. Network communication is not recovered and the WS-POS provider session time-out value is exceeded.		The WS-POS service consumer re-executes the **setEventResponse** and the provider session time-out value is exceeded indicating the network communication has not recovered. The WS-POS service consumer determines that the session with the WS-POS service provider has been lost. The WS-POS service provider clears the session and releases the UnifiedPOS devices.

			When the network communication has recovered, the WS-POS service consumer reissues the **openSession** method call. Thereafter it will be the same as a normal sequence.
H)	The polling of the WS-POS service provider from the WS-POS service consumer concludes normally and the event response setting is received but the return for the **setEventResponse** does not reach the WS-POS service consumer. Network communication is not recovered and the WS-POS provider session time-out value is exceeded.		The WS-POS service consumer re-executes the **setEventResponse** and the provider session time-out value is exceeded indicating the network communication has not recovered. The WS-POS service consumer determines that the session with the WS-POS service provider has been lost. The WS-POS service provider clears the session and releases the UnifiedPOS devices. When the network communication has recovered, the WS-POS service consumer re-executes the **openSession** method call. Thereafter it will be the same as a normal sequence.

2.4.15.1 Event Polling Return Value is Not Received and Session Does Not Time Out

In this scenario, when the event polling for the WS-POS service provider fails, the WS-POS service consumer re-executes the event polling.

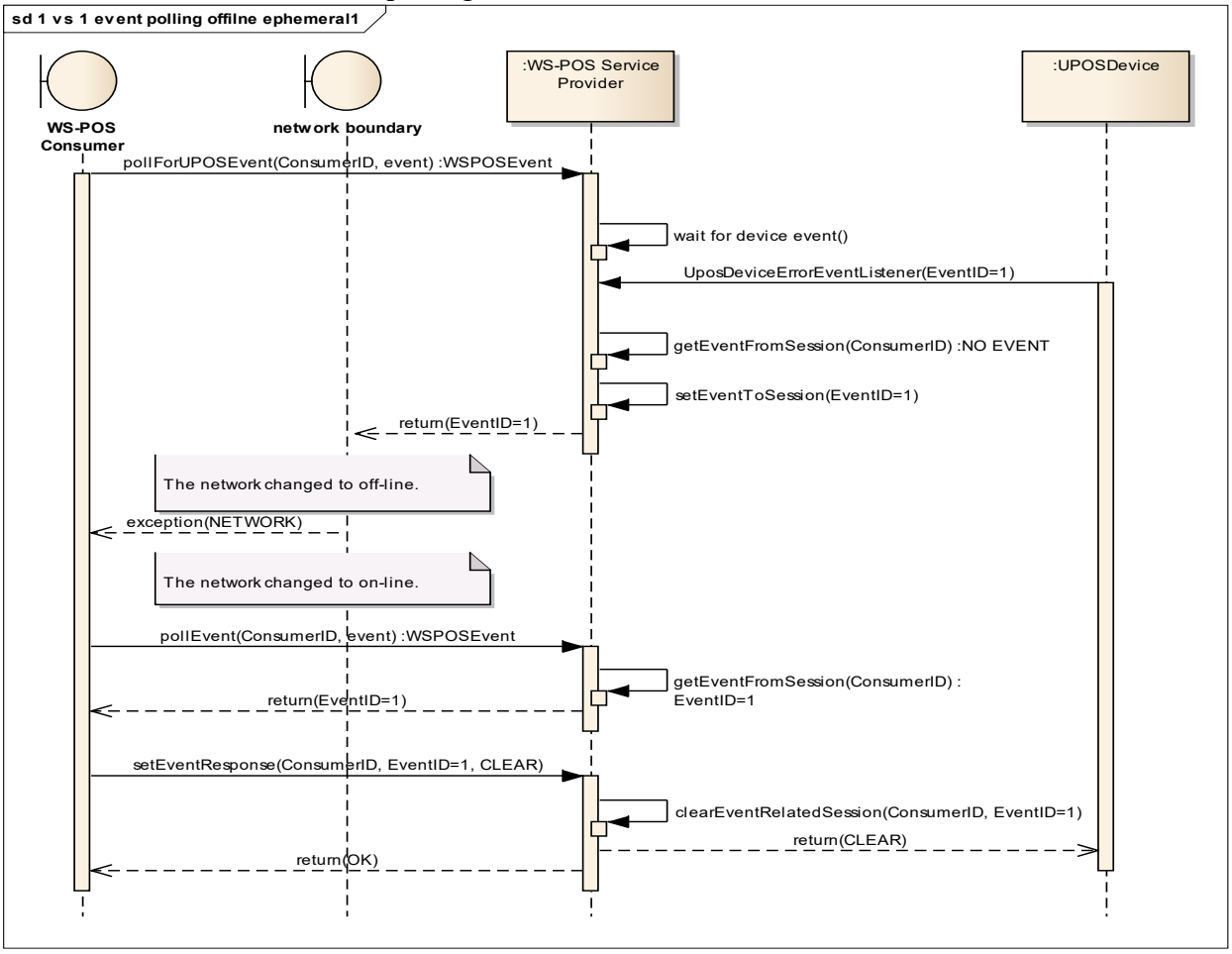


Figure 26: Offline Shorter than Session Time Out in Event Polling 1

The WS-POS consumer is able to receive events through re-polling after Network communication has been recovered in order for the WS-POS provider to hold events from POS devices until the **setEventResponse** succeeds.

2.4.15.2 SetEventResponse Return Value is Not Received and Session Does Not Time Out.

In this scenario, the network communication may disconnect when the **setEventResponse** is returned. The WS-POS service consumer re-tries sending the **setEventResponse** to the WS-POS service provider. At this time an **ILLEGAL** notification is returned in response to the initial event response after network communication has been recovered indicating that the event does not exist.

Although a network exception occurs for the WS-POS service consumer, the WS-POS service provider processes the event response and terminates the UnifiedPOS device event handler. As a result, as no event which requires a response exists, the event response after Network communication has been recovered causes an ILLEGAL notification to be returned.

Figure 27 illustrates this scenario.

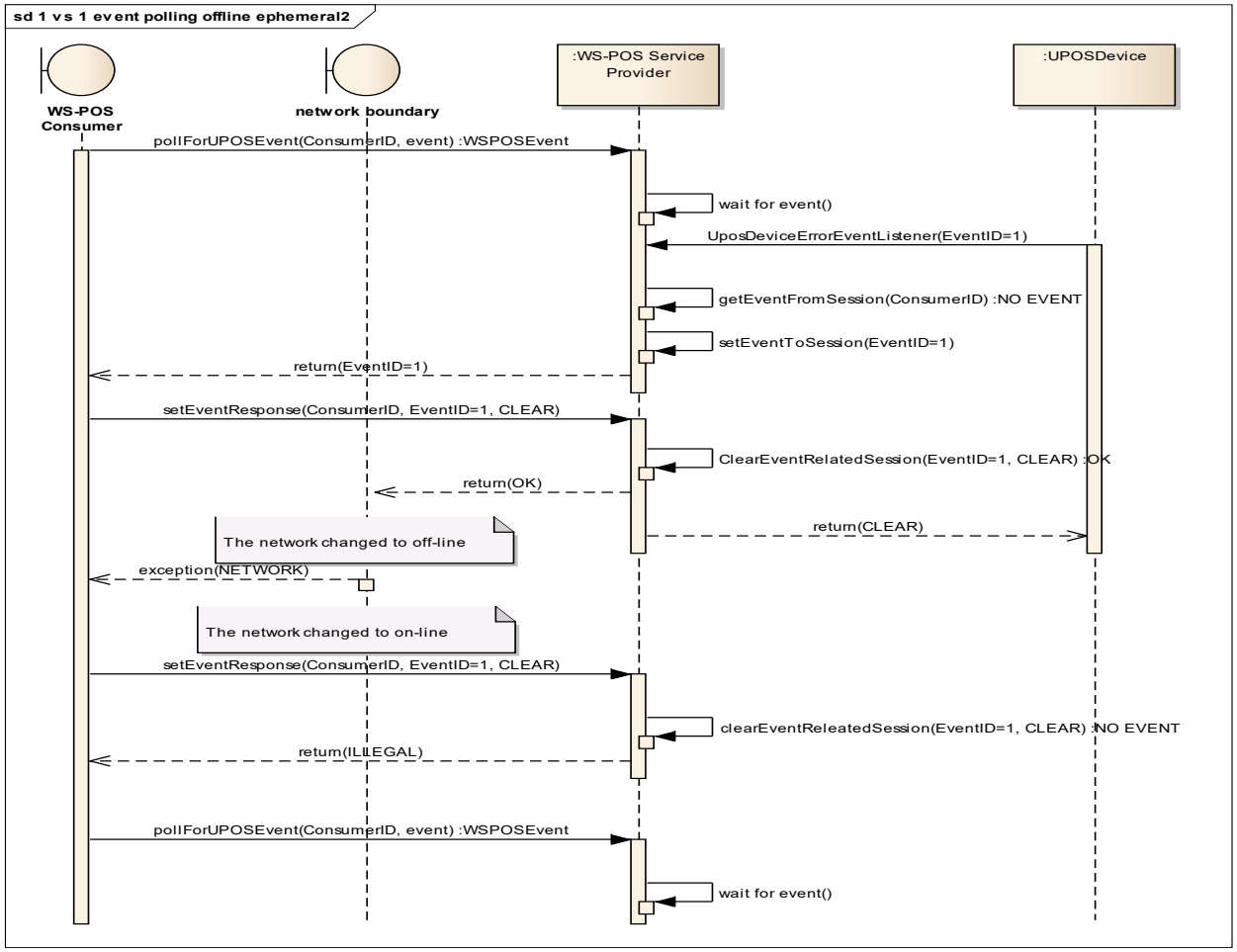


Figure 27: Offline Shorter than Session Time Out in Event Polling 2

2.4.15.3 Event Polling Is Not Received and Session Times Out

In this scenario, the network communication has not been recovered after the provider session time-out value has been exceeded. The WS-POS service provider cancels the event along with the session. When there is an **ErrorEvent**, a CLEAR operation is assigned in the response to the UnifiedPOS device.

In this case, the WS-POS service consumer is unable to receive the **ErrorEvent**. As the WS-POS service provider is clearing the session, even if Network communication is recovered, the WS-POS service consumer must re-execute an **openSession** method call.

Figure 28 illustrates this scenario.

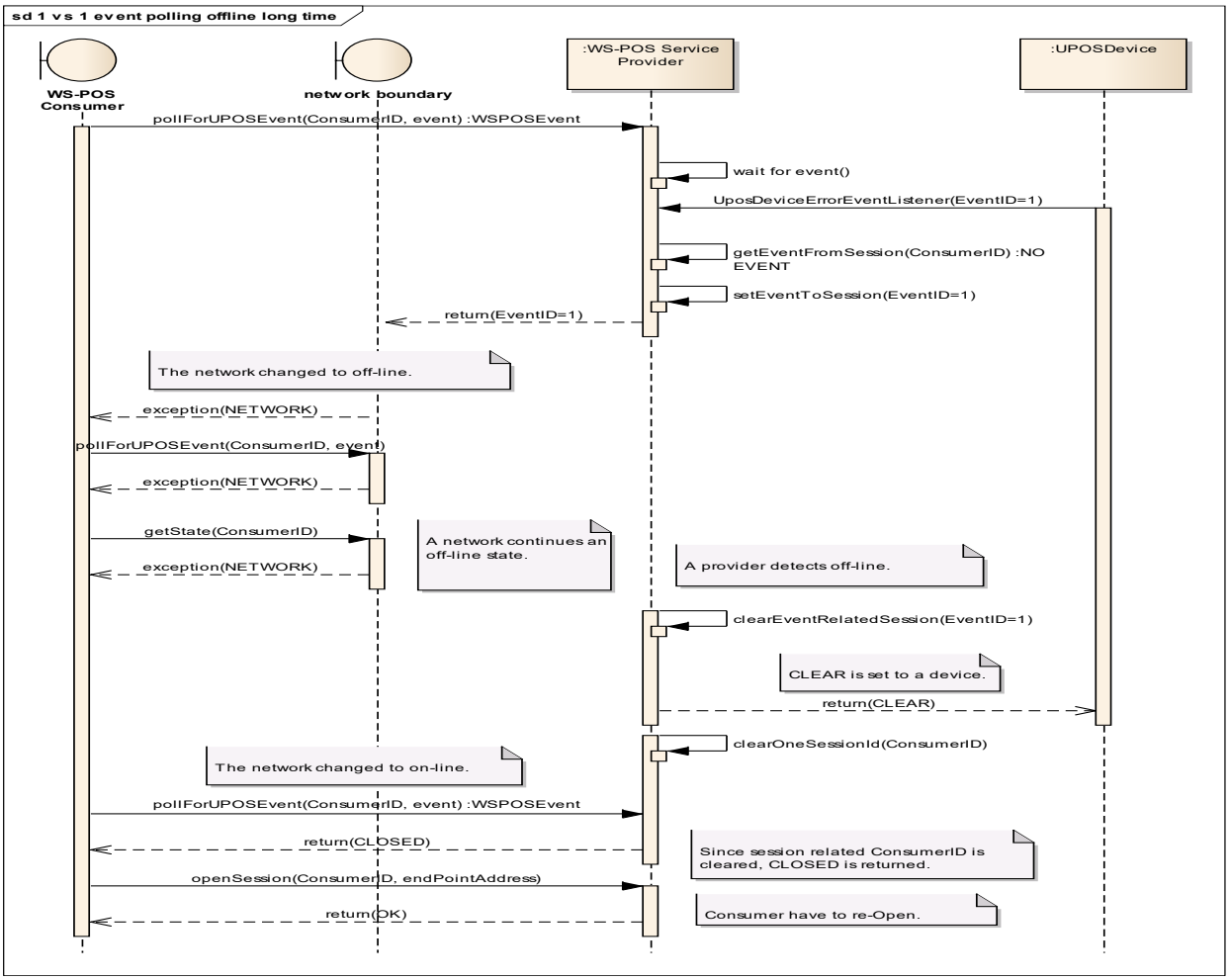


Figure 28: Offline Longer than Session Time Out in Event Polling

2.4.16 WS-POS Method References (UPOS UML Style) *Updated in Version 1.2*

The following are available Session level methods.

2.4.16.1 generateConsumerID Method

Syntax **generateConsumerID(out consumerID: *string*):**
void { raises-exception }

Parameter	Description
consumerID	The Unique ID to identify a WS-POS service consumer to a WS-POS Service Provider.

Remarks Generates a *consumerID* which is "guaranteed" to be unique; suggestion is to use an algorithm for a GUID or a UUID.

Errors An Exception may be thrown when this method is invoked.

Value	Meaning
E_FAILURE	The WS-POS Service Provider is not able to successfully calculate a *consumerID* .

2.4.16.2 openSession Method

Syntax **openSession(consumerID: string, consumerEventEndPoint: string): void {raises-exception}**

Parameter	Description
consumerID	The Unique ID to identify a WS-POS service consumer to a WS-POS Service Provider.
consumerEventEndPoint	The Service end point of WS-POS service consumer for WS-POS service provider to use in notifying an event. When this parameter is set to a NULL, the WS-POS service provider does not send an event. The WS-POS service consumer should retrieve a UnifiedPOS event by calling the **pollForUPOSEvent** method.

Remarks Establishes a session between WS-POS service consumer and WS-POS service provider.

Errors An Exception may be thrown when this method is invoked.

Value	Meaning
E_ILLEGAL	A session is open that has the same *consumerID*.

2.4.16.3 closeSession Method

Syntax **closeSession(consumerID: *string*):**
void {raises-exception}

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS Service Provider.

Remarks Terminates a session between a WS-POS service consumer and a WS-POS service provider.

Errors An Exception may be thrown when this method is invoked.

Value	Meaning
E_ILLEGAL	An **openSession** method has not been established. An invalid WS-POS *consumerID* has been passed. There is no WS-POS *consumerID* specified.

2.4.16.4 getProviderSessionTimeout Method

Syntax **getProviderSessionTimeout(consumerID: *string*, out timeout : *int32*): void {raises-exception}**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS Service Provider.
timeout	The value the WS-POS Service Provider is using for session response failure that the WS-POS Service Consumer can monitor network connection

Remarks This method returns WS-POS session timeout value in seconds, which is managed by WS-POS service provider.

Errors An Exception may be thrown when this method is invoked.

Value	Meaning
E_ILLEGAL	An **openSession** method has not been established. An invalid WS-POS *consumerID* has been passed. There is no WS-POS *consumerID* specified.

2.4.16.5 keepAlive Method

Syntax **keepAlive (consumerID: *string*):**
void {raises-exception}

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS Service Provider.

Remarks To keep the session between WS-POS service consumer and WS-POS service provider alive, a WS-POS service consumer should regularly call this method. A WS-POS service consumer must call the **keepAlive** method within a shorter time interval than the time out value which is returned by **getProviderSessionTimeout** method.

WS-POS service provider terminates the session when the WS-POS service provider does not receive a **keepAlive** method call within the timeout value returned by the **getProviderSessionTimeout** method.

To reconnect a session, the **openSession** method must be called.

Errors An Exception may be thrown when this method is invoked.

Value	Meaning
E_ILLEGAL	An **openSession** method has not been established. An invalid WS-POS *consumerID* has been passed. There is no WS-POS *consumerID* specified.

2.4.16.6 pollForUPOSEvent Method

Syntax **pollForUPOSEvent (consumerID: *string*, out WSPOSEvent: *string*): void {raises-exception}**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.
WSPOSEvent	The specific event as defined in UnifiedPOS

Remarks A WS-POS service consumer calls this method to retrieve a UnifiedPOS event occurrence. This can be any of the events defined in UnifiedPOS. If a UnifiedPOS event has occurred, the event notification is returned to WS-POS service consumer. Refer to section "2.4.17" for the details of WSPOSEvent.

If the WS-POS service consumer specifies a NULL in the *consumerEventEndPoint* parameter of **openSession** method, the WS-POS service consumer can call the **pollForUPOSEvent** method.

The **pollForUPOSEvent** method can be called after a **openSession** method call. However, the **pollForUPOSEvent** method will always return an **E_TIMEOUT** if no event has occurred before an **openDevice** method call.

Errors An Exception may be thrown when this method is invoked.
Some possible values of the exception's *ErrorCode* are:

Value	Meaning
E_ILLEGAL	An **openSession** method has not been established. An invalid WS-POS *consumerID* has been passed. There is no WS-POS *consumerID* specified. **pollForUPOSEvent** method is called after **openSession** method without setting a NULL in consumerEventEndPoint parameter.
E_TIMEOUT	No event has occurred within the event polling timeout value which is set by the WS-POS service provider.

2.4.16.7 setEventResponse Method

Syntax **setEventResponse (consumerID: *string*, WSPOSEventResponse: *string*): void {raises-exception}**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.
WSPOSEventResponse	Information about the event retrieved by **pollForUPOSEvent** method

Remarks After the WS-POS service consumer receives an event from the **pollForUPOSEvent** method, the WS-POS service consumer will call this method to inform the WS-POS service provider that it has finished its response to the event.

Refer to section "2.4.17" for the details of WSPOSEventResponse.

If the WS-POS service consumer does not call this method before the WS-POS session timeout expires, the WS-POS service provider will begin an error recovery process.

Errors An Exception may be thrown when this method is invoked.

Value	Meaning
E_ILLEGAL	An **openSession** method has not been established. An invalid WS-POS *consumerID* has been passed. There is no WS-POS *consumerID* specified.

2.4.16.8 getWSPOSVersion Method

Syntax **getWSPOSVersion (out WSPOSVersion: *string*): void {raises-exception}**

Parameter	Description
WSPOSVersion	The version that the WS-POS service provider supports.

Remarks This method returns the WS-POS version implemented by the WS-POS service provider.

2.4.16.9 getEncryptedClaimedConsumerID Method *Added in Version 1.3*

Syntax **getEncryptedClaimedConsumerID (out encryptedClaimedConsumerID: *string*): void {raises-exception}**

Parameter	Description
encryptedClaimedConsumerID	Encrypted (Hash) value of consumerID of WS-POS service consumer that has been successfully claimed by WS-POS service provider.

Remarks

A WS-POS service consumer calls this method to retrieve the encrypted value of the consumerID of another WS-POS service consumer that currently has claimed the WS-POS service provider. The WS-POS service provider searches for a session under its management, identifies the consumerID of the session that currently has claimed the device, and returns the session's encrypted value. If no WS-POS service consumer has claimed the device, the WS-POS service provider returns a null.

In POS system environments where UnifiedPOS applications coexist in the same physical hardware terminal and where the WS-POS service provider is running, the POS device is considered as a shared device; a UnifiedPOS application claimed the device, WS-POS service provider returns null, but the device is claimed.

2.4.16.10 setEventRequestProperties Method *Added in Version 1.3*

Syntax **setEventRequestProperties(parameters: *List*): void {raises-exception}**

Remarks

Parameter	Description
parameters	List of property names to be notified in DataContainedEvent. The property names are the properties that a WS-POS service consumer retrieves by DataContainedEvent. Those properties are the one that are set in UnifiedPOS DataEvent. Ex: ScanDataType (Scanner), Track1Data(MSR)…

WS-POS service consumer specifies the names of the properties to be notified in DataContainedEvent to WS-POS service provider. If the name of the property is specified with this method, DataContainedEvent is notified instead DataEvent.

If the parameter is null or empty, DataEvent is notified instead DataContainedEvent.

The following are available Device level methods.

2.4.16.11 openDevice Method

Syntax **openDevice(consumerID: *string*):**
void {raises-exception}

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.

Remarks A WS-POS service consumer calls this method to open the UnifiedPOS device. The WS-POS service provider instantiates and opens the UnifiedPOS device.

Errors An Exception may be thrown when this method is invoked.
Some possible values of the exception's *ErrorCode* are:

Value	Meaning
E_ILLEGAL	There is no session specified by consumerID. Or the device is already opened. The Unified POS Control is already open.
E_NOEXIST	The logical name which service provider uses is not registered as a UnifiedPOS driver in the node where the WS-POS service provider is running.
E_NOSERVICE	Could not establish a connection to the corresponding Unified POS Service.

2.4.16.12 closeDevice Method

Syntax **closeDevice(consumerID: *string*):**
void {raises-exception}

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS Service Provider.

Remarks A WS-POS service consumer calls this method to close the UnifiedPOS device. The WS-POS service provider closes its UnifiedPOS service object and terminates the device instance.

Errors An Exception may be thrown when this method is invoked.
Some possible values of the exception's *ErrorCode* are:

Value	Meaning
E_ILLEGAL	There is no session specified by *consumerID*.
E_CLOSED	Already closed.

2.4.16.13 getBinaryConversion Method *Added in Version 1.3*

Syntax **getBinaryConversion(consumerID: *string*, out binaryConversion, *int*):**
void {raises-exception}

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS Service Provider.
binaryConversion	Refer to UnifiedPOS specification for BinaryConversion.

Remarks A WS-POS service consumer calls this method to get the BinaryConversion property of the UnifiedPOS device.

Refer to UnifiedPOS specification for BinaryConversion.

Errors An Exception may be thrown when this method is invoked.
Some possible values of the exception's *ErrorCode* are:

Value	Meaning
E_ILLEGAL	There is no session specified by *consumerID*.
E_CLOSED	Already closed.

2.4.16.14 setBinaryConversion Method *Added in Version 1.3*

Syntax **setBinaryConversion(consumerID: *string*, binaryConversion, *int*): void {raises-exception}**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS Service Provider.
binaryConversion	Refer to UnifiedPOS specification for BinaryConversion.

Remarks A WS-POS service consumer calls this method to set the BinaryConversion property of the UnifiedPOS device.

Refer to UnifiedPOS specification for BinaryConversion.

Errors An Exception may be thrown when this method is invoked.
Some possible values of the exception's *ErrorCode* are:

Value	Meaning
E_ILLEGAL	There is no session specified by *consumerID*.
E_CLOSED	Already closed.

2.4.17 WSPOS Event and WSPOS Event Response *Added in Version 1.2*

2.4.17.1 WSPOSEvent

The WSPOSEvent that is returned by the **pollForUPOSEvent** method call reflects a UnifiedPOS device class pending event state.

For example, the **pollForUPOSEvent** method of POSPrinter class returns POSPrinterEvent type, and PollForUPOSEvent of Scanner returns the ScannerEvent type.

Following table shows event types in the WSPOSEvent.

UPOS Event	Description
DataEvent	This is a **DataEvent** defined in the UnifiedPOS specification. A WS-POS service provider sets relevant properties and data fields prior to when the **DataEvent** occurs, and transfers to a WS-POS service consumer. When other event occurs, a NULL is set.
StatusUpdateEvent	This is a **StatusUpdateEvent** defined in UnifiedPOS specification. A WS-POS service provider sets relevant properties and data fields prior to when the **StatusUpdateEvent** occurs, and transfers to a WS-POS service consumer. When other event occurs, a NULL is set.
DirectIOEvent	This is a **DirectIOEvent** defined in UnifiedPOS specification. A WS-POS service provider sets relevant properties and data fields prior to when the **DirectIOEvent** occurs, and transfers to a WS-POS service consumer. When other event occurs, a NULL is set.
OutputCompleteEvent	This is a **OutputCompleteEvent** defined in UnifiedPOS specification. A WS-POS service provider sets relevant properties and data fields prior to when the **OutputCompleteEvent** occurs, and transfers to a WS-POS service consumer. When other event occurs, a NULL is set.
ErrorEvent	This is a **ErrorEven**t defined in UnifiedPOS specification. A WS-POS service provider sets relevant properties and data fields prior to when the **ErrorEvent** occurs, and transfers to a WS-POS service consumer. When other event occurs, a NULL is set.

*Note: With UnifiedPOS 1.14 an additional Event, "**TransitionEvent**" is defined for the Electronic Value Reader/Writer Device class. It is handled in a similar manner as these events shown here.*

The WSPOSEvent represents event types that a UnifiedPOS device class returns. i.e. EventType returned varies depending on device class. As an example, POSPrinterEvent for output device POSPrinter is a combination **of StatusUpdateEvent**, **DirectIOEvent, OutputCompleteEvent** and **ErrorEvent**. DataEvent is not defined for POSPrinter device class.

Figure 29: WSPOS POS Printer Event

The ScannerEvent of input device Scanner is a combination of **DataEvent**, **StatusUpdateEvent**, **DirectIOEvent** and **ErrorEvent**. OutputCompleteEvent is not defined in Scanner device class.

Figure 30: WSPOS - Scanner Event

Refer to UnifiedPOS specification for details of **DataEvent**, **StatusUpdateEvent**, **DirctIOEvent**, **OutputCompleteEvent** and **ErrorEvent**.

*Note: With UnifiedPOS 1.14 an additional Event, "**TransitionEvent**" is defined for the Electronic Value Reader/Writer Device class. It is handled in a similar manner as these events shown here.*

2.4.17.2 WSPOSEventResponse

WSPOSEventResponse that is set to **setEventResponse** method by a WS-POS service consumer is an EventResponse type of a device class in actual implementation.

For example, the **setEventResponse** method for POSPrinter device should be set POSPrinterResponse as WSPOSEventResponse. The **setEventResponse** method for Scanner device should be set ScannerResponse as WSPOSEventResponse.

Following table shows event types in the WSPOSEventResponse.

Type	Desciption	Related Event
DataEventResponse	When DataEvent in WSPOSEvent that is returned by **pollForUPOSEvent** method is not a NULL, a WS-POS service consumer should set a valid value to DataEventResponse as WSPOSEventResponse, then notify a WS-POS service provider. DataEventResponse has an EventID item. The WS-POS service consumer should set EventID value in DataEventResponse, which is set in **DataEvent** as a return value in **pollForUPOSEvent** method call.	**DataEvent**
StatusUpdateEventRespose	When **StatusUpdateEvent** in WSPOSEvent that is returned by **pollForUPOSEvent** method is not a NULL, a WS-POS service consumer should set a valid value to StatusUpdateEventResponse as WSPOSEventResponse, then notify a WS-POS service provider. StatusUpdateEventResponse has an EventID item. The WS-POS service consumer should set EventID value in StatusUpdateEventResponse,	**StatusUpdateEvent**

	which is set in **StatusUpdateEvent** as a return value in **pollForUPOSEvent** method call.	
DirectIOEventResponse	When **DirectIOEvent** in WSPOSEvent that is returned by **pollForUPOSEvent** method is not a NULL, a WS-POS service consumer should set a valid value to DirectIOEventResponse as WSPOSEventResponse, then notify a WS-POS service provider. DirectIOEventResponse has an EventID item. The WS-POS service consumer should set EventID value in DirectIOEventResponse, which is set in **DirectIOEvent** as a return value in **pollForUPOSEvent** method call. A WS-POS service consumer can set any data in DirectIOData in DirectIOEventResponse.	**DirectIOEvent**
OutputCompleteEventResponse	When **OutputCompleteEvent** in WSPOSEvent that is returned by **pollForUPOSEvent** method is not a NULL, a WS-POS service consumer should set a valid value to OutputCompleteEventResponse as WSPOSEventResponse, then notify a WS-POS service provider. OutputCompleteEventResponse has an EventID item. The WS-POS service consumer should set EventID value in OutputCompleteEventResponse, which is set in **OutputCompleteEvent** as a return value in **pollForUPOSEvent** method call.	**OutputCompleteEvent**
ErrorEventResponse	When ErrorEvent in WSPOSEvent that is returned by	**ErrorEvent**

	pollForUPOSEvent method is not a NULL, a WS-POS service consumer should set a valid value to ErrorEventResponse as WSPOSEventResponse, then notify a WS-POS service provider. ErrorEventResponse has an EventID item and an ErrorResponse item. The WS-POS service consumer should set EventID value in ErrorEventResponse, which is set in **ErrorEvent** as a return value in **pollForUPOSEvent** method call. The WS-POS service consumer should also set either "Clear, "ContinueInput" or "Retry" in ErrorResponse.	

The WSPOSEventResponse represents EventResponse types that a UnifiedPOS device class receives. They are some of 5 event responses - DataEventResponse, StatuUpdateEventResponse, DirectIOEventResponse, OutputCompleteEventResponse and/or ErrorEventResponse which is related to **DataEvent**, **StatusUpdateEvent**, **DirectIOEvent**, **OutputCompleteEvent** and **ErrorEvent** respectively.

*Note: With UnifiedPOS 1.14 an additional Event, "**TransitionEvent**" is defined for the Electronic Value Reader/Writer Device class. It is handled in a similar manner as these events shown here.*

For example, POSPrinterEventResponse of output device POSPrinter is a combination of StatusUpdateEventResponse, DirectIOEventResponse, OutputCompleteEventResponse and ErrorEventResponse.

Figure 31: WSPOS Event Response

ScannerEventResponse of input device Scanner is a combination of DataEventResponse, StatusUpdateEventResponse, DirectIOEventResponse and ErrorEventResponse.

Figure 32: WSPOS - Event Response

2.4.18 WS-POS Event Reference in Bi-Directional Communication

Updated in Version1.3

A method reference of events used in WS-POS bi-directional communication is described in this section.

These methods are published by WS-POS service consumer, and WS-POS service provider calls them. Therefore, ***inout*** declaration of a parameter in this reference means that there is a case where the WS-POS service consumer passes information to the WS-POS service provider such as data or obj in **DirectIOEvent**, and errorResponse in **ErrorEvent**.
Depending on the device class, some of the event methods below may not be called by a WS-POS service provider. For examples, the input device "Scanner" never calls **outputCompleteEvent**; the output device "POSPrinter" never calls **DataEvent**.

In Polling event handling, these methods are never called by the WS-POS service provider. The WS-POS service consumer should retrieve an event by using the **pollForUPOSEvent** method, and respond to the event by using the **setEventResponse** method.

2.4.18.1 dataEvent Method

Syntax **dataEvent (consumerID: *string*, source: *string*, eventID: *int32*, timeStamp: *DateTime*, status: *int32*): void**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.
source	UnifiedPOS device class
eventID	Event ID notified by UnifiedPOS device
timeStamp	Time stamp information notified by UnifiedPOS device
status	Status information notified by UnifiedPOS device UnifiedPOS Specification: The input status with its value dependent upon the device category; it may describe the type or qualities of the input data.

Remarks After a bi-directional session is established, if a **DataEvent** is fired by a UnifiedPOS device which was opened by a WS-POS service consumer request through a WS-POS service provider, the WS-POS service provider passes this method back to the WS-POS service consumer. Refer to UnifiedPOS specification for event handling.

2.4.18.2 directIOEvent Method

Syntax **dataEvent (consumerID: *string*, source: *string*, eventID: *int32*, timeStamp: *DateTime*, eventNumber: *int32*, inout data: *int32*, inout obj: *object*) : void**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.
source	UnifiedPOS device class
eventID	Event ID notified by UnifiedPOS device
timeStamp	Time stamp information notified by UnifiedPOS device
eventNumber	EventNumber information notified by UnifiedPOS device UnifiedPOS Specification: Event number whose specific values are assigned by the UnifiedPOS Service.
data	Numeric data information notified by UnifiedPOS device

	UnifiedPOS Specification: Additional numeric data. Specific values vary by the EventNumber and the UnifiedPOS Service. This attribute is settable.
obj	Object information notified by UnifiedPOS device UnifiedPOS Specification: Additional data whose usage varies by the EventNumber and the UnifiedPOS Service. This attribute is settable.

Remarks After a bi-directional session is established, if a **DirectIOEvent** is fired by a UnifiedPOS device which was opened by a WS-POS service consumer request through a WS-POS service provider, the WS-POS service provider passes this method back to the WS-POS service consumer. Refer to UnifiedPOS specification for event handling.

2.4.18.3 errorEvent Method

Syntax **errorEvent (consumerID: *string*, source: *string*, eventID: *int32*, timeStamp: *DateTime*, errorCode: *int32*, errorCodeExtended: *int32*, errorLocus: *int32*, inout errorResponse: *int32*) : void**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.
source	UnifiedPOS device class
eventID	Event ID notified by UnifiedPOS device
timeStamp	Time stamp information notified by UnifiedPOS device
errorCode	ErrorCode notified by UnifiedPOS device UnifiedPOS Specification: Error Code causing the error event.
errorCodeExtended	ErrorCodeExtended notified by UPOS device UnifiedPOS Specification: Extended Error Code causing the error event. These values are device category specific.
errorLocus	ErrorLocus notified by UPOS device UPOS Specification: Location of the error.
errorResponse	ErrorResponse notified by UPOS device UPOS Specification: Error response, whose default value may be overridden by the application (i.e., this property is settable).

Remarks After a bi-directional session is established, if an **ErrorEvent** is fired by a UPnifiedOS device which was opened by a WS-POS service consumer request through a WS-POS service provider, the WS-POS service provider passes this method back to the WS-POS service consumer. Refer to UnifiedPOS specification for event handling.

2.4.18.4 outputCompleteEvent Method

Syntax **outputCompleteEvent (consumerID: *string,* source: *string,* eventID: *int32,* timeStamp: *DateTime,* outputID: *int32*) : void**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.
source	UnifiedPOS device class
eventID	Event ID notified by UnifiedPOS device
timeStamp	Time stamp information notified by UnifiedPOS device
outputID	OutputID notified by UnifiedPOS device UnifiedPOS Specification: The ID number of the asynchronous output request that is complete.

Remarks After a bi-directional session is established, if an **OutputCompleteEvent** is fired by a UnifiedPOS device which was opened by a WS-POS service consumer request through a WS-POS service provider, the WS-POS service provider passes this method back to the WS-POS service consumer. Refer to UnifiedPOS specification for event handling.

2.4.18.5 statusUpdateEvent Method

Syntax **statusUpdateEvent (consumerID: *string,* source: *string,* eventID: *int32,* timeStamp: *DateTime,* status: *int32*): void**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.
source	UnifiedPOS device class
eventID	Event ID notified by UnifiedPOS device
timeStamp	Time stamp information notified by UnifiedPOS device
status	Status information notified by UnifiedPOS device UnifiedPOS Specification: Device category-specific status, describing the type of status change.

Remarks After a bi-directional session is established, if a **StatusUpdateEven**t is fired by a UnifiedPOS device which was opened by a WS-POS service consumer request through a WS-POS service provider, the WS-POS service provider passes this method back to the WS-POS service consumer. Refer to UnifiedPOS specification for event handling.

2.4.18.6 dataContainedEvent Method *Added in Version 1.3*

Syntax **dataContainedEvent (consumerID: *string,* source: *string,* eventID: *int32,* timeStamp: *DateTime,* status: *int32,* parameters: *KeyValuePairList*): void**

Parameter	Description
consumerID	The unique ID to identify a WS-POS service consumer to a WS-POS service provider.
source	UnifiedPOS device class
eventID	Event ID notified by UnifiedPOS device
timeStamp	Time stamp information notified by UnifiedPOS device
status	Status information notified by UnifiedPOS device UnifiedPOS Specification: Device category-specific status, describing the type of status change.
parameters	The Key Value Pair list, specified by the setEventRequestProperties method.

Remarks After establishing a bi-directional communication session where the WS-POS service consumer has specified the properties that will be delivered as part of an event (by

previously calling the setEventRequestProperties method) and when the UnifiedPOS device which was opened by WS-POS service provider fires a DataEvent, the WS-POS service provider calls this method to deliver the notification and data associated with the event to the WS-POS service consumer. Refer to UnifiedPOS specification for more information on event handling.

2.4.19 Modifications to XMLPOS *Updated in Version 1.3*

The following modifications are needed to support the XMLPOS schemas which are defined by ARTS UnifiedPOS 1.14.1 and used within WS-POS 1.3.

- Convert ARTSBinary to String data
- Add getEncryptedClaimedConsumerID method
- Add the option of using DataEventResponse which allows for event data to be sent as defined by the setEventResponse
- Add dataContainedEvent event
- Add key-value pair type to store the property values returned with a dataContainedEvent event response
- Change type of DataEvent that is the return value of pollForUPOSEvent method
- Add the getBinaryConversion method and setBinaryConversion method

The definition of ARTSBinary in ARTS XML Best Practices limits the available characters for a string type.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:simpleType name="ARTSBinary">
    <xs:restriction base="xs:string">
      <xs:pattern value="(\\u[0-9 | a-f]{4,4})*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

However, since it is difficult for this restriction to be supported by all the possible programming language contracts in a unified way, the WS-POS specification treats ARTSBinary as a string type. This insures the compatibility of the contracts and the XML schema.

2.4.19.1 Addition and Deletion of Methods

The following methods are added for each UnifiedPOS 1.14.1 XMLPOS WSDL and xsd.

Method	Description
generateConsumerID	Generates Consumer ID as GUID or UUID
openSession	Starts a WS-POS Session (Device is not opened)
closeSession	Closes a WS-POS Session
getProviderSessionTimeout	Returns Session Timeout value which WS-POS service provider sets

keepAlive	WS-POS service consumer regularly calls this method to keep session active
pollForUPOSEvent	WS-POS service consumer calls this method to check if an event has occurred
setEventResponse	Selects the WS-POS service consumer response to the event retrieved by the pollForUPOSEvent method
getWSPOSVersion	Returns WS-POS version information for the WS-POS service provider
openDevice	Opens a UnifiedPOS device
closeDevice	Closes a UnifiedPOS device
getEncryptedClaimedConsumerID	*Added in WS-POS Version 1.3* Retrieves the encrypted value of the consumerID of the session that has been successfully claimed by WS-POS service provider.
setEventRequestProperties	*Added in WS-POS Version 1.3* The WS-POS service consumer uses this method to specify the names of properties that will be returned by the WS-POS service provider DataContainedEvent.
getBinaryConversion	*Added in WS-POS Version 1.3* The WS-POS service consumer uses this method to get the BinaryConversion property of UnifiedPOS Device. Refer to UnifiedPOS specification for BinaryConversion.
setBinaryConversion	*Added in WS-POS Version 1.3* The WS-POS service consumer uses this method to set the BinaryConversion property of UnifiedPOS Device. Refer to UnifiedPOS specification for BinaryConversion.

The following methods are deleted for each UnifiedPOS 1.14 XMLPOS WSDL and xsd.

Method	Description
open	This UnifiedPOS method is deleted and replaced by using the **openSession** and **openDevice** methods.
close	This UnifiedPOS method is deleted and replaced by using the **closeSession** and **closeDevice** methods.

The following event is added for each UnifiedPOS 1.14.1 XMLPOS WSDL and xsd. *Added in Version 1.3*

Event	Description
dataContainedEvent	When a DataEvent is fired by a UnifiedPOS device, the event notification will return Key-Value pair data, defined by the **setEventRequestProperties** method, to the WS-POS service consumer.

2.4.19.2 Addition of ConsumerID Parameter

In WS-POS 1.2, the ConsumerID is used to identify WS-POS service consumer. It is mandatory that the ConsumerID is a statistically unique identifier. The entity can be generated using any algorithm that statistically guarantees that it is unique. Typical algorithms are the UUID or GUID and examples for generation can be found in RFC4122 by "The Internet Society", http://www.ietf.org/rfc/rfc4122.txt . Since the ConsumerID is statistically unique, it only has to be generated once if the WS-POS service consumer has the ability to permanently store and use it for all future transactions. If the WS-POS service consumer does not have a unique ConsumerID, WS-POS 1.2 requires that the WS-POS service provider have an algorithm that will create a ConsumerID and return it back to the WS-POS service consumer to utilize.

The ConsumerID parameter is the first parameter to all of methods (including Getter and Setter properties). All of xsd schemas of XMLPOS listed in UnifedPOS 1.14.1 must be modified to account for this requirement.

The following examples are used to illustrate this requirement.

UnifiedPOS 1.14 .1XML POS GetDeviceEnabled(no parameter) description -> XML POS for WS-POS 1.2 – red characters are added

```
<xs:element name="GetDeviceEnabled">
 <xs:complexType>
  <xs:sequence />
 </xs:complexType>
</xs:element>
<xs:element name="GetDeviceEnabledResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element
     minOccurs="0" name="GetDeviceEnabledResult"
    type="xs:boolean" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

->

```
<xs:element name="GetDeviceEnabled">
 <xs:complexType>
   <xs:element
     minOccurs="0"  name="ConsumerID" nillable="true"
     type="xs:string" />
  </xs:complexType>
</xs:element>
<xs:element name="GetDeviceEnabledResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element
    minOccurs="0" name="GetDeviceEnabledResult"
    type="xs:boolean" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

UnifiedPOS 1.14.1 XML POS SetDeviceEnabled(with parameter) description -> XML POS for WS-POS 1.2 – red characters are added

```
<xs:element name="SetDeviceEnabled">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0"
      name="DeviceEnabled" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="SetDeviceEnabledResponse">
  <xs:complexType>
    <xs:sequence />
  </xs:complexType>
</xs:element>
```

->

```
<xs:element name="SetDeviceEnabled">
  <xs:complexType>
    <xs:sequence>
    <xs:element
        minOccurs="0"  name="ConsumerID" nillable="true"
        type="xs:string" />
      <xs:element minOccurs="0"
      name="DeviceEnabled" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="SetDeviceEnabledResponse">
  <xs:complexType>
    <xs:sequence />
  </xs:complexType>
</xs:element>
```

All of UnifiedPOS 1.14.1 peripheral xsd's need to be similarly modified.

2.4.20 File Path for a Method Call *Added in Version 1.2*

Some of the UnifiedPOS methods, such as SetBitmap of POSPrinter class, have a parameter for the file path name. In WS-POS, this is not appropriate, because a WS-POS service provider and a WS-POS service consumer may run on different devices. To support the parameter in WS-POS context without breaking UnifiedPOS API semantics, a path name parameter is defined as follows:

- A path name parameter which is passed from a WS-POS service consumer to a WS-POS service provider should be a URI (Uniform Resource Indicator).
- A WS-POS service provider should support at least “http: schema”.
- A WS-POS service provider is recommended to support “https:, file:” and “data: schemas” as an option.
- When a WS-POS service consumer passes “file: URI” to a WS-POS service provider, the URI is a file on the device where the WS-POS service provider is running.
- When a WS-POS service provider receives “file: URI”, the URI format depends on how a WS-POS service provider implemented on a device. Therefore, it is out of scope of the WS-POS specification.

The following steps represent a typical use case.

1. A WS-POS service consumer copies a file to a file sharing service such as Skydrive before the WS-POS service consumer calls a WS-POS service provider.

NOTE: The WS-POS service provider does not support authentication to access to the file sharing service with the URI.

2. The WS-POS service consumer passes the URI to the WS-POS service provider as a request parameter.
3. The WS-POS service provider retrieves the resource that is specified by the URI.
4. The WS-POS service provider may cache the resource on its local device to improve a response for subsequent access requests.
5. The WS-POS service provider passes the resource retrieved to the device service. When the device service requires a file resource on the local storage, the WS-POS service provider creates a temporary file, and then passes the file path name to the temporary file. The temporary file is deleted when the device method call is completed.
6. The WS-POS service provider returns the results of the device service method call.

2.4.20.1 Security Considerations

The following are some examples on the importance for the file content to be protected and secured:

- A WS-POS service provider must validate the file which is passed by a URI, especially, in the case where a firmware update file for a peripheral device is transferred.
- If the URI is file: schema, the resource must be an accessible directory or file validated by a WS-POS service consumer.
- If the URI is http: schema, the host in URI must be listed in an authorized list of WS-POS service providers.
- If the URI is data: schema, the data length must be validated as the original file size.

2.4.21 Retrieving consumerID that has been successfully claimed

Added in Version 1.3

When more than one WS-POS service consumer is connected to a WS-POS service provider, there is a case where a WS-POS service consumer may want to know which WS-POS service consumer is successful in claim the UnifiedPOS device from the WS-POS service provider. In WS-POS version 1.3, a function which retrieves encrypted value of consumerID of WS-POS service consumer that has successfully claimed is added as getEncryptedClaimedConsumerID method.

2.4.21.1 Security Considerations

To be able to obtain another WS-POS service consumer's consumerID makes any WS-POS service consumer can access illegally to the device that is claimed by the WS-POS service consumer by using the obtained consumerID.

For example, when consumerID of WS-POS service consumer (A) which claims MSR is obtained by another WS-POS service consumer (B), the WS-POS service consumer (B) can read out TrackData of MSR by using the obtained consumerID.

To avoid this, a method (getEncryptedCalimedConsumerID) that retrieves encrypted value of consumerID instead retrieving consumerID of another WS-POS service consumer is defined.

In this way, even if the encrypted value is leaked, the security is high since it is difficult to decode consumerID from the encrypted value.

Figure 33 below shows a potential security breach because WS-POS service consumer (B) can successfully read out the TrackData of MSR which is claimed by WS-POS service consumer (A), since the consumer ID is not encrypted.

Figure 33: Get Claimed Consumer ID

Figure 34 shows no potential security breach because WS-POS service consumer (B) cannot successfully read out the TrackData of MSR which is claimed by WS-POS service consumer (A), since the consumer ID is encrypted.

Figure 34: Get Encrypted Claimed Consumer ID

The following sequence diagram shows an example on how to identify the WS-POS service consumer A who keeps claiming the service for whatever the reason. In this example consumer C who wants to use the WS-POS service, can externally make a request to the system administrator who has the decryption key to identify consumer A as the one who has the blocking claim. The system administrator can then communicate with consumer A and initiate a clear blocked claim policy to free up the WS-POS service.

There are a few alternative ways to solve this problem. One is to allow the WS-POS service provider to monitor and control the blocking WS-POS service consumer activities with timeouts. However, this alternative resolution method makes the system architecture more complex with possible additional methods and properties necessary. Therefore, in WS-POS Version 1.3, only getEncryptedClaimedConsumerID method is added to solve the problem because of simplicity in implementation.

2.5 Status, State Model and Exceptions

As shown below, there are some commonly used enumerations, events, and properties in which the status, error code, and state model are set.

2.5.1 StatusUpdateEvent

StatusUpdateEvent is an event generated when the state unique to the specific class and status variables are changed.

2.5.2 ControlState

ControlState is an enumeration to store the current state. Possible values are below.

Closed
Idle
Busy
Error

2.5.3 Exceptions

There is the possibility that all WS-POS service method calls throw **UposException** when they fail. **UposException** is defined in each device under the namespace http://www.nrf-arts.org/UnifiedPOS/ by the XMLPOS schema of ARTS.

2.5.4 Public Properties

Name	Description
ErrorCode	**ErrorCode** is an error code which represents the cause of the error exception and is defined by an enumeration.
ErrorCodeExtended	**ErrorCodeExtended** is an extension error code which represents the cause of the error exception. Values unique to the service can be stored in this code.

2.6 Shared Device Model

The shared device model of WS-POS supports devices that can be partially or completely shared by multiple applications (WS-POS service consumer) as well as devices that can only be exclusively used by one WS-POS service consumer simultaneously. All WS-POS service providers are open to one or more applications. However, some behaviors that can be executed in a WS-POS service provider limit applications that acquire access rights to a device to only one application.

2.6.1 Exclusive Use Device

The most common device types are “Exclusive Use Devices” and one such example is the POS printer. This device can be used simultaneously by one application only (WS-POS service consumer) due to its physical characteristics and characteristics of operation. The WS-POS service consumer must call the **Claim** method and enable exclusive access before most of the methods, properties, and events become effective. If a method is called or a property is set before the exclusive access right is acquired, an illegal error occurs.

If two closely related WS-POS service consumers would like to use an exclusive use device by sharing, there is a way where one WS-POS service consumer has exclusive access to the device for a certain period of time only and releases the exclusive access right later in order to allow the other WS-POS service consumer to also use the device.

If the **Claim** method is called again, the settable characteristics of devices are restored to the state when the **Release** method is called. Examples of restoration include the brightness of the line display, the read track of magnetic stripe readers, and the number of characters per POS printer line. The State characteristics like the sensor property of the POSPrinter are not restored and are set to the current state instead.

2.6.2 Sharable Devices

Some devices are sharable devices. One example is key lock. For sharable devices, the properties can be accessed by the calling of its methods by multiple applications (WS-POS service consumers). In addition, events are notified to all applications to which the device is open. However, a WS-POS service consumer with exclusive access to the sharable device can limit the access to part of the methods and properties and only this WS-POS service consumer can be notified of events.

2.7 Event Messages

The event notifies the application (WS-POS service consumer) of various behavior and changes to the device or removal of the device. There are the following five events.

Event	Description
DataEvent	Input data is stored in device class unique properties.
ErrorEvent	An error occurred during an event-driven input or asynchronous output.
StatusUpdateEvent	State change of the device is reported.
OutputCompleteEvent	Asynchronous output was normally completed.
DirectIOEvent	This event is defined by a service provider and items that cannot be covered by this specification will be provided.

The WS-POS service provider queues events when events are generated. Queued events are notified (delivered) to the application (WS-POS service consumer) when they are ready to be notified. The causes of delays in the sending and receiving of events include the following.

- The WS-POS service consumer has set the **FreezeEvents** property to true.
- The event type is a **DataEvent** or input **ErrorEvent**, and the **DataEventEnabled** property is false.

For terminology concerning the notification of events, the following terms except for "Event" in this paragraph are not used separately, and are published for reference.

Reference: The following terms concerning events are used in this specification.

Queue	When a service decides the necessity of event notification (Fire) to the WS-POS service consumer, the service packs the event in an internal event queue.
Deliver	When the event queue is not empty and all requirements of the first event in the queue are met, this event is removed from the queue and the event notification request for the application is executed.
Fire	It can be said that Fire is the combination of queuing (Queue) and notification (Delivery). However, Fire sometimes is used in broad terms and refers to only one of these steps. The meaning must be identified by the context.

Regulations concerning the management of the event queues are as follows.

- While the device is in the “enable” state, the WS-POS service provider is only queuing (Queue) new events.

- There is a possibility that the WS-POS service provider conducts notification (Delivery) of queued events until the WS-POS service consumer calls the **release** method (exclusive use device) or the **closeDevice** method (all devices). Note that all remaining events are deleted when either the **release** or the **closeDevice** methods are executed.

- In the input device, the **ClearInput** method clears data and the input error event. Within the event handler, the WS-POS service consumer can access properties and call methods. However, the WS-POS service consumer must not call the **release** method and the **closeDevice** method within the event handler since the **release** method must close event handling (including the thread notifying (Delivering) the event) and **closeDevice** must close the event handling before it returns.

2.8 Input Model

The WS-POS input model supports event-driven input. In event-driven input, input data can be received after **DeviceEnabled** is set to true. The received data is queued as **DataEvent** events for notification to the application (WS-POS service consumer) when the requirements are met. If the **AutoDisable** property is true when data is received, the WS-POS service provider sets **DeviceEnabled** to false and automatically disables the device. As a result, further queuing of input by the service is prevented and the device is physically disabled (if possible).

When the WS-POS service consumer is ready to receive input from the device, the WS-POS service consumer sets the **DataEventEnabled** property to true. When the input is received, the WS-POS service provider queues and notifies the **DataEvent** event. (If the input is already queued, the **DataEvent** event is notified.) This event may include input status information. Immediately before the event is notified, the WS-POS service provider stores input data and other information in the device unique property based on necessity.

The WS-POS service provider makes further data events impossible by internally setting the **DataEventEnabled** property to false immediately before the event is notified. As a result, any subsequent input data is queued by the service while the WS-POS service consumer is processing the current input and related properties. When the WS-POS service consumer completes processing of the current input and is prepared to receive any subsequent data, it sets **DataEventEnabled** to true to restart the notification of events.

If the input device is an exclusive use device, the WS-POS service consumer must both acquire and enable the exclusive access right of the device prior to start of input reading by the device.

In case of sharable input devices, the WS-POS service consumer must open and enable the device prior to the start of input reading by the device. The WS-POS service consumer must call the **claim** method and request exclusive access to the device before the WS-POS service provider sends data using the **DataEvent** event. When event-driven input is received, if the WS-POS service consumer does not have the exclusive access right to the device, the input is queued until the exclusive access right of the device is acquired (and the **DataEventEnabled** property becomes true). As a result, multiple WS-POS service consumers can share the device in order and input focus can be given and received effectively between WS-POS service consumers.

If the WS-POS service provider finds an error during the input processing of data from event-driven input, the WS-POS service provider changes its **State** property to Error and queues one to two **ErrorEvent** events in order to warn the WS-POS service consumer of the error state. Since this event is not notified until the **DataEventEnabled** property becomes true, the WS-POS service consumer can conduct processing in order. Error events are notified accompanied by data showing the next error position.

InputData:

> **InputData** is queued if an error occurs while one or more **DataEvent** events are queued. This event is queued before all other **DataEvent** events. Due to this event, the WS-POS service consumer can clear the input immediately or has the option of warning users of the error and processing the buffered input data.

The latter case is effective for scanners. Since error warnings can be quickly sent to users, subsequent items are not be scanned until the error is removed. Previously scanned items can be normally processed before error recovery is executed.

Input:

Input is notified when an error occurs and there is no usable data. (In general implementations, this event is set at the end of the event queue.) If part of the input data has already been queued when the error occurs, the **ErrorEvent** event whose error position is **InputData** is first queued and notified. This error event is notified after all **DataEvent** events are notified. (When the "**InputData**" event is notified and the event handler of WS-POS service consumer responds "clear", this "Input" event is not notified.)

In any of the following conditions, the WS-POS service provider ends the Error state when the WS-POS service consumer:

- "Reads" the Input data associated with the InputData **ErrorEvent**.
- "Responds" to the **InputData ErrorEvent** event with an **ErrorResponse** of "clear".
- "Calls the **ClearInput** method.

The method for starting an event-driven input is dependent upon the WS-POS service consumer making an input data request. Subsequent input data is not accepted (normally) until the method is issued again. This will re-initialize the input buffer after the WS-POS service consumer has received the previous input data from the device. Examples include the operation of reading MICR data and signature capture data. No new data will be sent from these devices until the service consumer is ready to accept new data. This type of event-driven input is defined as "Asynchronous input".

The **DataCount** property is used to keep track of the number of **DataEvent** events enqueued by the WS-POS service provider.

All input data that is queued by the WS-POS service provider can be deleted by calling the **ClearInput** method. The **ClearInput** method can be called after **openDevice** in sharable devices and after **claim** in exclusive use devices.

For the general (Asynchronous) event-driven input model, specific methods, device classes and properties are not required to be defined to return the input data. There are some devices that require methods to return input data and populate specific properties. An example of this is the key lock device. Its locked or unlocked key position status is held in a defined property value determined by an input request. This type of an input is referred to as "Synchronous input".

2.9 Output Model

The output of WS-POS consists of two output types of *Synchronous* and *Asynchronous*. Each device class may support either, both or neither of these output types.

2.9.1 Synchronous Output

This output type is effective for when it is necessary to output data to a POS device promptly and the application cannot continue until the data has been transferred. The advantage of synchronous output lies in its simplicity.

The application (WS-POS service consumer) calls a class unique method which handles the data output. The WS-POS service provider does not return control back to the application until all the output data has been transferred or an error condition exists.

2.9.2 Asynchronous Output

If the POS device can only handle transfer of data at a low-speed or large blocks of data are required to be transferred, this output type is effective. The application (WS-POS service consumer) sets up a buffer of data that needs to be output to the POS device, executes an asynchronous method call, returns to continue with other computing tasks, and receives a completion or error event when the data has been transferred. Since the WS-POS service consumer can execute other processing when the device is executing the output, the advantage of this output type lies in the ability of the application to multi-task, potentially speeding up POS services.

Further details to the process are as follows:

- The WS-POS service consumer calls a class unique method and starts output. The service buffers the request within program memory, and the service sends it to the physical device immediately as soon as reception processing becomes possible in the physical device. Then, the service sets the identifier of this request in the **OutputId** property and returns back to the application as soon as possible. When the device normally completes the request, the WS-POS service provider notifies the application using an **OutputCompleteEvent** event. The parameter of this event is the **OutputId** of the last successful output complete request.

- If an error occurs when an asynchronous data output is being executed, the **ErrorEvent** event is fired. The event handler of the WS-POS service consumer can retry or clear the output of any outstanding transfer request of queued data. The service remains in the Error state during processing of the **ErrorEvent** event. (Note: If the cause of the error has not been removed when the service returns from an **ErrorEvent** event, another **ErrorEvent** event will fire if the POS device is still in an error state.)

- Asynchronous output is queued based on FIFO buffer technique. All buffered output (including all asynchronous outputs) can be deleted by calling the **ClearOutput** method. The **OutputCompleteEvent** event is not generated as a result of calling the **ClearOutput** method. Calling the **ClearOutput** method should terminate all remaining data from being sent, if possible.

2.10 Device Power Reporting Model

In an application (WS-POS service consumer), the state of the POS device power must be available for a query. This information is conveyed by the **PowerState** enumeration.

Note: *This model does not target notification of the power supply state ("battery on" and "battery low ", etc.) of PCs and the POS terminal main power unit. Notification of this information is managed by utilizing the UnifiedPOS "POS Power" device specification.*

2.10.1 Model

The power state of WS-POS devices may be denoted using one of the following conditions.

- **Online** The POS device is in the "*Power On and Ready*" mode. This means that it is available for use.

- **Off** The POS device is in the "*Power Off*" mode or it is not connected to the POS system. It implies that the POS Device is not available for use.

- **Offline** The POS device is in the "*Not Ready*" mode or cannot respond even though the POS device is in the "*Power On*" mode. It implies that the POS Device is not available for use. It may be necessary to push a physical button to place the POS device into an "*Online*" mode. While in the "*Offline"* state the POS device may not be able to respond to requests from an application.

- **OffOffline** The POS device is in an undeterminable *"Off"* or *"Offline*" state. This means the POS device is in a "*Power Off"* or "*Not-Ready"* mode; the device service is not able to distinguish between these two states.

The power state notification only functions for exclusive POS devices if the application is in the programmatic "*open, claim*, *enable"* condition.

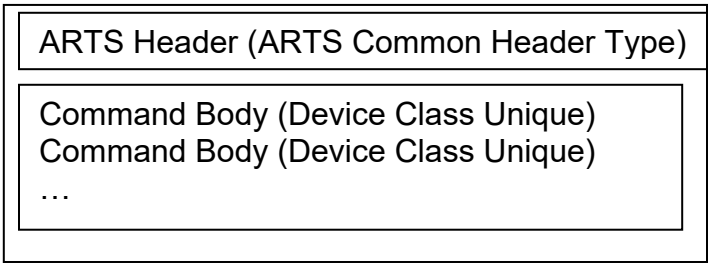
Note: Programmatic vs. Physical (Hardware) *"Enable/Disable"* State

The context of where the terms "enable/disable" determines the meaning and usually is completely different. In WS-POS application service definition, "enable" or "disable" is a programmatic logical state. In the WS-POS Power Supply Notification model, physical (Hardware) context, "enable" or "disable" refers to the physical state of the POS device. Depending on the POS device, even if the programmatic state is logically "enable", the physical (Hardware) may be in the "disable" state, physically "offline". Conversely, if the programmatic state is logically "disable", the physical (Hardware) state may be Online ("enable") because it is in the power on and ready mode.

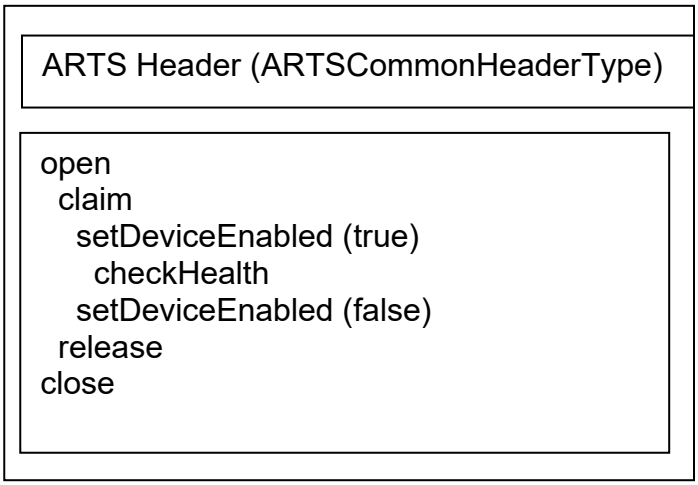
Regardless of the physical (Hardware) state, WS-POS Power State Notification is only available to the application when in the programmatic "enable" state. This restriction is necessary since services can generally only communicate with devices during a programmatic "enable state". Even when a POS device is physically "Offline", the service may try and fail a programmatic device "enable". However, once the POS device becomes "Online" (physical "enable") and the programmatic state is "enable", the service will not automatically change the state of the programmatic "enable" to "disable" even if the power state changes.

2.11 ARTS XMLPOS Command Set

In ARTS XMLPOS, the Command Set to transmit a series of processing for methods and properties by batch message is defined.
Command Set is comprised of the following.



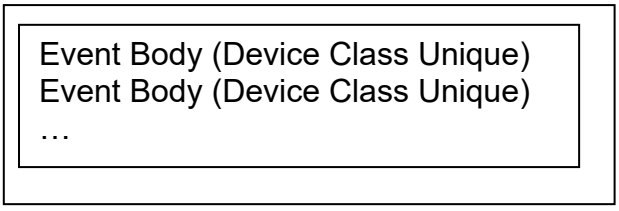
For example, in the batch processing of **open, claim, enabled, checkHealth, disable, release and close**,



it is possible to batch up the commands and transmit them in one data exchange. As a result, the message granularity between WS-POS service consumers and WS-POS service providers is flexible and can be appropriately adjusted as the circumstances require.

2.12 ARTS XMLPOS Event Set

In ARTS XMLPOS, Event Set to notify the service consumer of multiple kinds of events collectively is defined.
Event Set is comprised of the following.



2.13 ARTS XMLPOS Schema

WS-POS uses the ARTS XMLPOS schema for the definition of a device class. The ARTS XMLPOS schema is provided in an "xsd" file.

The corresponding schema file names for device classes are as follows.

Schema	Description
<Device Class Name> <Version>.xsd	Schema defining the properties/methods of a device class
<Device Class Name> CommandSet <Version>.xsd	Schema defining Command Set of a device class
<Device Class Name> Event <Version>.xsd	Schema defining the events of a device class
<Device Class Name> EventSet <Version>.xsd	Schema defining Event Set of a device class

Note: *In this example, the* Version *is a character string that starts with prefix "V" and consists of the "major version", the "minor version", and the "bug fix" version using period "." delimiters. (Example: V1.14)*
Please refer to ARTS SOA Best Practices Technical Report for further version information. The ARTS XMLPOS schema adopts the version of UnifiedPOS.

2.14 WS-POS WSDL

The WSDL file group below defines the WSDL of the WS-POS service in accordance with the ARTS XMLPOS schema. The WS-POS WSDL's are based upon the corresponding version of the WS-POS standard and the ARTS XMLPOS as defined in the corresponding version of ARTS UnifiedPOS standard.

Although the following WSDL are compatible with the ARTS XMLPOS schema, since binding information is also described in WSDL, the binding information must be changed in accordance with that used by the WS-POS service provider for actual use. The binding information of WSDL included in this WSDL file is the placeholder.

WSDL	Description
<Device Class Name> <Version>.wsdl	WSDL describing the properties /methods of the device class
<Device Class Name> CommandSet <Version>.wsdl	WSDL describing Command Set of the device class
<Device Class Name> Event <Version>.wsdl	WSDL describing the events of the device class
<Device Class Name> EventSet <Version>.wsdl	WSDL describing Event Set of the device class

Note: *In this example, the* Version *is a character string that starts with prefix "V" and consists of the "major version",*
the "minor version", and the "bug fix" version using period "." delimiters. (Example: V1.14)
Please refer to ARTS SOA Best Practices Technical Report for further version information.
The ARTS XMLPOS schema adopts the version of Unified POS.

2.14.1 Web Service Description Language (WSDL)

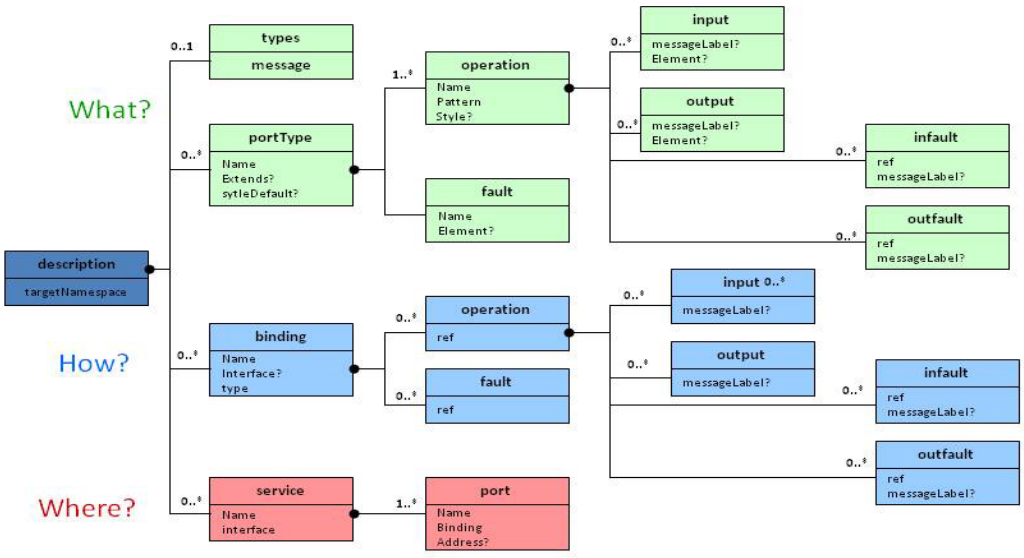


Figure 35: WSDL Overview

WS-POS employs WSDL 1.1, as explained in WS-I basic profile version 1.2, as the core service description facility.

Id	Name	Description
SD001	WSDL	A device's web services MUST be conformant to WSDLs as specified in the "Service Description" section of this specification
SD002	WSA-Action	To be conformant, the receiving Web Services MUST discard SOAP messages that do not contain WS-Addressing Action headers

2.14.2 WS-POS WSDL Interoperability and Development Platform Requirements to Ensure Equivalent Code Conversion

In order to ensure hardware and software interoperability, a WS-POS WSDL must be identical whether it was created from Java code or .NET code.

In practice this means a WCF contract is used to process Windows .NET code to produce a WS-POS WSDL. Also a JAX-WS contract is used to process Java code to create a WS-POS WSDL. Both of these WS-POS WSDLs must be identical. This is the only way of ensuring that WS-POS interoperability can be guaranteed regardless of the development language used to create the WSDL.

The following diagram graphically shows this relationship and the importance of equivalent WS-POS WSDL creation.

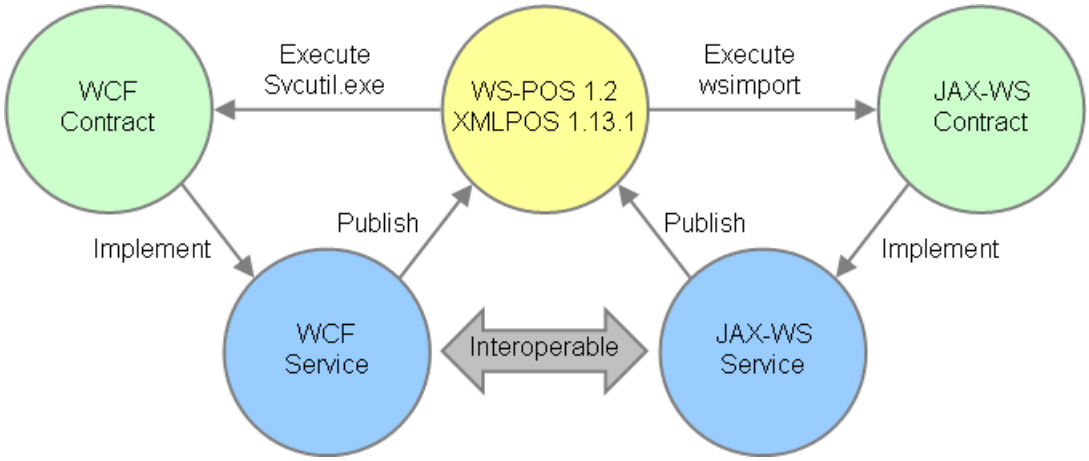


Figure 36: Interoperability and Platform Equivalence to the Code Conversion

The WS-POS standard uses the current UnifiedPOS standard XMLPOS messages to define UnifiedPOS classes that are mapped using the ARTS defined WSDLs. This ensures consistent WSDL mapping of the UnifiedPOS Properties, Methods, and Events for the peripheral devices. In addition, the UnifiedPOS Device list of arguments, return values and exceptions are respectively mapped by the WSDL to the Request message, Response message, and Fault message.

Finally, the UnifiedPOS data types are mapped to equivalent XML schema data types.

Figure 37 shows the relationships of these mappings.

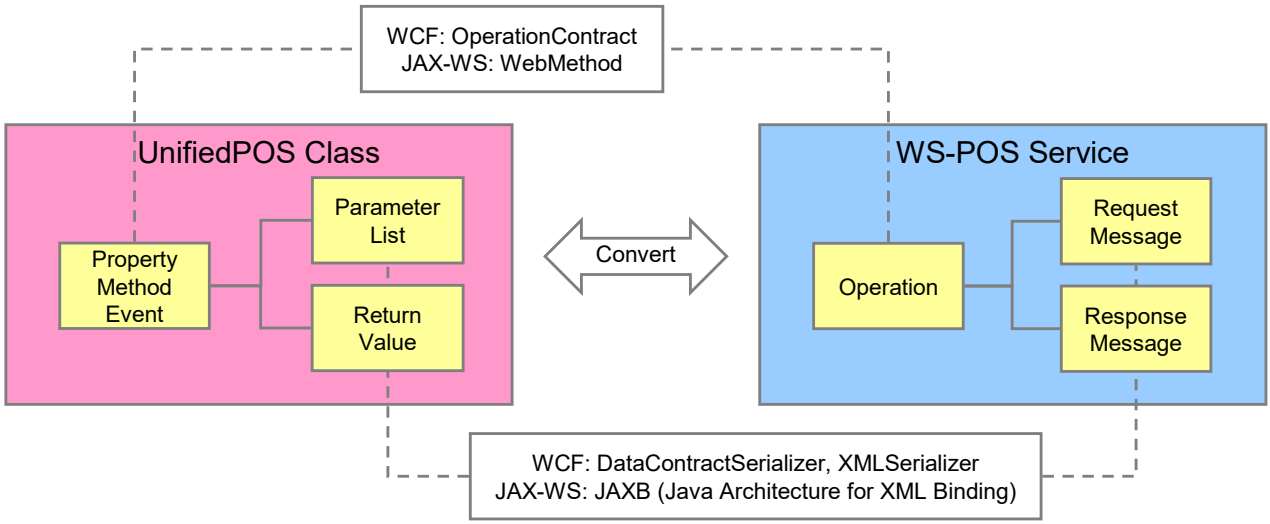


Figure 37: UnifiedPOS and WSDL Mapping

2.14.3 WSDL Documents Provided By ARTS

For each device type, ARTS provides two categories of WSDLs: one to describe the services that a producing device implements, and another to describe the services a consuming application for that device type needs to implement. Each of these provided WSDLs will contain one or more port types with sets of operations, all the messages for each operation, and a corresponding SOAP binding for each port type.

The ARTS WSDL files are part of the WS-POS 1.3 download package. The most recent WS-POS documentation and WSDL package zip and tar files can be found under the WS-POS section at http://www.omg.org/retail/unified-pos.htm.

2.14.4 WSDL Documents to Be Created by Implementers

For each device or client wishing to implement WS-POS, it is necessary to create a new WSDL document that imports the corresponding ARTS WSDL. The implementer then creates a service with an endpoint address and ports referring to one or more of the bindings in the ARTS WSDL.

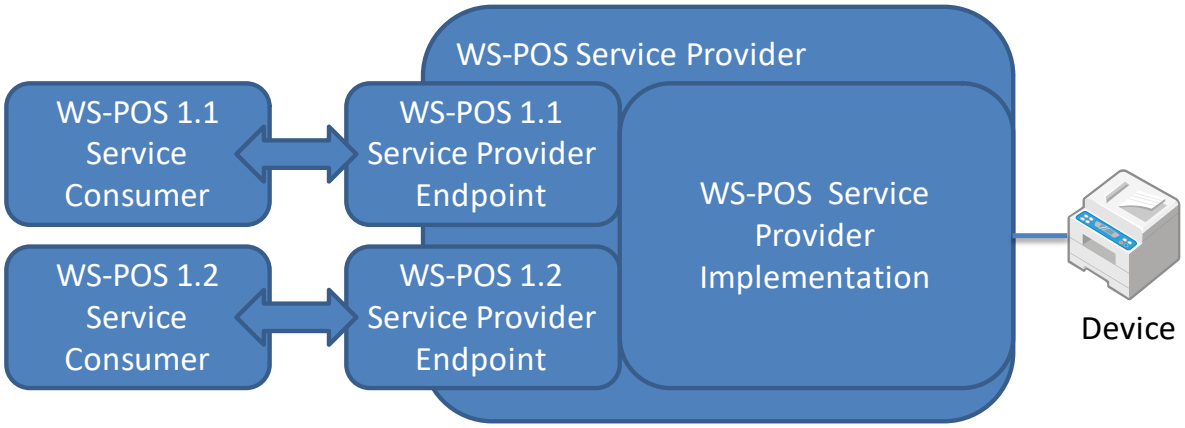
Id	Name	Description
SD003	ARTS-Binding	An implementing service must refer to the bindings from the corresponding provided ARTS WSDL

2.15 Backward Compatibility *Added in Version 1.2*

2.15.1 How to use the WSDLs from Version 1.1 and from Version 1.2

The WSDLs defined and created for WS-POS 1.1 and WS-POS 1.2 are not the same, due to functional changes made in version 1.2 of the WS-POS standard. This prevents interoperability between WS-POS 1.1 consumers/providers and WS-POS 1.2 consumers/providers. It is highly recommended that a user migrates to WS-POS 1.2 compliant service consumers and providers because of the increased functionality added in the WS-POS 1.2 version.

To be clear, the version of the WS-POS service consumer and the WS-POS service provider must be the same.



2.16 Security

2.16.1 Description

Security is a key focus of the entire ARTS distributed peripheral system. In a consumer system of this nature, which exchanges sensitive information such as credit card numbers and details of purchases, it is imperative that the exchanged messages are secure. Measures must be taken to ensure that devices are authenticated, that authorizations are assured, that transmitted information is always safe from snooping and alteration and that the transmitted information goes to the proper endpoint.

2.16.2 Transport Layer Security *Changed in Version 1.3*

In this WS-POS technical specification, it is recommended to guarantee network transport layer security, but is not defined specific method to secure the transport layer, so that a system integrator can apply appropriate security standards in system design depending on user's requirement.

2.16.3 Device Authentication with UDDI(Deprecated) *Changed in Version 1.3*

In this WS-POS technical specification, using UDDI for device authentication is not mandatory. When UDDI is used for device authentication, it must follow the description below.

The UDDI server must have a method to verify that device keys are acceptable. This may take the form of a preloaded table, verification with some outside authority, etc.

Devices must provide signatures for their interactions with UDDI. The UDDI server must then use these signatures to validate that the device is allowed to use the server.

Id	Name	Description
SE001	DeviceAuthUDDI	UDDI must reject all unsigned messages from devices
SE002	DeviceAuthUDDIFail	UDDI must reject all messages that are signed by unauthorized devices.

2.16.4 Client Authentication with UDDI(Deprecated) *Changed in Version 1.3*

In this WS-POS technical specification, using UDDI for client authentication is not mandatory. When UDDI is used for client authentication, it must follow the description below

The UDDI server must have a method to verify that client keys are acceptable. This may take the form of a preloaded table, verification from some outside authority, etc.

Clients must provide signatures for their interactions with UDDI. The UDDI server must then use these signatures to validate that the client is allowed to use the server.

Id	Name	Description
SE003	ClientAuthUDDI	UDDI must reject all unsigned messages from clients
SE004	ClientAuthUDDIFail	UDDI must reject all messages that are signed by unauthorized clients.

2.16.5 Client Authentication with Device

When interacting with devices, the client must sign all of its messages. The first message, which opens the device, will contain an endpoint reference that the device will send events to. The device must store an (endpoint reference, signature) name-value pair for the client in order to validate future communications. When the client sends messages to the device, the device will use the endpoint reference in the "<from>" WS-Addressing field along with the included signature to validate the message. When opening and closing the device, the device must validate that the endpoint reference in the "<from>" WS-Addressing field matches the endpoint reference in the SOAP-Body.

Id	Name	Description
SE005	ClientAuthDevice	Device must reject all unsigned messages from clients
SE006	FromSIGMatch	Device must reject all messages whose <from> element in the header does not match the EndpointReference stored for that signature
SE007	FromEPRMatch	Device must reject all open and close messages in which the EndpointReference in the SOAP body's <in> doesn't match the <from> in the SOAP header

Note: A client only has the ability to register its own endpoint reference; no attempt to define how a client can register any other endpoint references has been made. This is a known endpoint reference registration limitation issue currently in all forms of web services event registration.

2.17 XML Payload

While it is described in the WSDLs, it is worth noting that the XML payload for all the SOAP messages is almost entirely specified using XMLPOS, with the exception of the callback methods on the clients.

3. GENERAL FLOW

3.1 General WS-POS Flows

This section covers the prescribed methods for using the WS-POS stack.

3.1.1 Device service initializes

1. Setup device - Address, capabilities, naming, location of UDDI
2. Register with UDDI – authenticate itself, address, capabilities, naming
    a. Message signed with device private key
    b. UDDI validates signature

3.1.2 Device shutdown

1. Device deletes itself from UDDI
2. Power off scanner device

3.1.3 Client startup

1. Setup client entity service - address, capabilities, naming, location of UDDI

3.1.4 Client interaction with devices

1. Query UDDI for list of device services – authenticate itself
    a. Message signed using Client private key
2. UDDI returns list of devices to client- Client chooses which device to connect to
3. Client entity service opens device service, registers for events (atomic)
    a. Message signed with Client private key
4. Device validates signature
5. Device stores endpoint reference/certificate pair for future verification
6. Client entity service – claim, enable device service
    a. both messages signed with Client private key
7. User interacts with device
    a. device service sends events to the client (Signed with Device Private key)
8. As client changes settings, settings are stored in endpoint reference/certificate table
    a. All messages to device are signed by Client
9. Client entity service disables, releases device
    a. Message signed with Client private key
10. Client entity service closes device
    a. Message signed with client private key
    b. Device validates client endpoint reference against signature

3.2 Simple Use Case

3.2.1 Client Entity Service Acquiring and Utilizing a Scanner Device Service

A client entity service used in a point-of-service solution allows the clerk to scan items. This service acquires and utilizes a scanner device service to input transaction item identification. The client entity service releases the scanner device service after the sales transaction completes.

3.2.1.1 SOA Main Flow Description

1. Setup UDDI – authentication parameters for scanner device service and client entity service
2. Power on the scanner device
3. Scanner device service initializes
    - Setup scanner - Address, capabilities, naming, location of UDDI
    - Register with UDDI – authenticate itself, provide implementation WSDL
        i. Message signed with Scanner private key
        ii. UDDI validates signature
4. The Client entity service is loaded and executed.
    - Setup client entity service - address, capabilities, naming, location of UDDI
    - Query UDDI for list of scanner device services – authenticate itself
        i. Message signed using Client private key
5. UDDI returns list of devices - Client entity service chooses which scanner to connect to
6. Client entity service opens scanner device service, registers for events (atomic)

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/"
standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa=”http://www.w3.org/2005/08/addressing” xmlns:ns3="http://www.nrf-
arts.org/UnifiedPOS/Scanner/">
  <soapenv:Header>
       <wsa:To>http://scanner.WS-POS-example.omg.org:8081/services/Scanner-WS</wsa:To>
       <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
       <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/Open</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
       <ns3:Open>
            <ns3:EndpointAddress>http://192.168.1.1:8082/axis2/services/Client-
WS</ns3:EndpointAddress>
       </ns3:Open>
  </soapenv:Body>
</soapenv:Envelope>
```

    - Message signed with Client private key

7. Scanner validates signature
   Scanner stores endpoint reference/certificate pair

8. Client entity service – claim, enable scanner device service (both messages signed with Client private key)

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/"
standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa=”http://www.w3.org/2005/08/addressing” xmlns:ns3="http://www.nrf-
arts.org/UnifiedPOS/Scanner/">
  <soapenv:Header>
       <wsa:To>http://scanner.WS-POS-example.omg.org:8081/services/Scanner-WS</wsa:To>
       <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
       <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/Claim</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
       <ns3:Claim>
            <ns3:Timeout>1000</ns3:Timeout>
       </ns3:Claim>
  </soapenv:Body>
</soapenv:Envelope>

<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/"
standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa=”http://www.w3.org/2005/08/addressing” xmlns:ns3="http://www.nrf-
arts.org/UnifiedPOS/Scanner/">
  <soapenv:Header>
       <wsa:To>http://scanner.WS-POS-example.omg.org:8081/services/Scanner-WS</wsa:To>
       <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
       <wsa:Action>http://www.nrf-
arts.org/UnifiedPOS/Scanner/SetDeviceEnabled</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
       <ns3:SetDeviceEnabled>
            <ns3:DeviceEnabled>true</ns3:DeviceEnabled>
       </ns3:SetDeviceEnabled>
  </soapenv:Body>
</soapenv:Envelope>
```

9. As client changes settings, settings are stored on a per-client basis
10. Clerk scans an item

11. Scanner device service sends WS-POS event to the client entity service (calling the client entity service callback method using XMLPOS) (Signed with Scanner Private key)

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/"
standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa=”http://www.w3.org/2005/08/addressing” xmlns:ns3=”http://www.nrf-
arts.org/UnifiedPOS/ScannerEvents/”>
  <soapenv:Header>
       <wsa:To>http://client.WS-POS-example.omg.org/services/Client-WS</wsa:To>
       <wsa:ReplyTo>
       <wsa:Address>http://www.w3.org/2005/08/addressing/none</wsa:Address>
       </wsa:ReplyTo>
       <wsa:MessageID>urn:uuid:548181C361F2A1EE5C1195691857095</wsa:MessageID>
       <wsa:Action>http://www.nrf-
arts.org/UnifiedPOS/ScannerEvents/DataEvent</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
       <ns3:DataEvent>
            <ns3:Source>http://scanner.WS-POS-example.nrf-arts.org/services/Scanner-
WS</ns3:Source>
            <ns3:EventID>1234</ns3:EventID>
            <ns3:TimeStamp>2011-01-09T12:34:56</ns3:TimeStamp>
            <ns3:Status>0</ns3:Status>
       </ns3:DataEvent>
  </soapenv:Body>
</soapenv:Envelope>
```

Client entity service receives the event

12. Client entity service disables, releases and closes scanner device

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/"
standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa=”http://www.w3.org/2005/08/addressing” xmlns:ns3="http://www.nrf-
arts.org/UnifiedPOS/Scanner/">
  <soapenv:Header>
       <wsa:To>http://scanner.WS-POS-example.omg.org/services/Scanner-WS</wsa:To>
       <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
       <wsa:Action>http://www.nrf-
arts.org/UnifiedPOS/Scanner/SetDeviceEnabled</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
       <ns3:SetDeviceEnabled>
       <ns3:DeviceEnabled>false</ns3:DeviceEnabled>
       </ns3:SetDeviceEnabled>
  </soapenv:Body>
</soapenv:Envelope>

<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/"
standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa=”http://www.w3.org/2005/08/addressing” xmlns:ns3="http://www.nrf-
arts.org/UnifiedPOS/Scanner/">
  <soapenv:Header>
       <wsa:To>http://scanner.WS-POS-example.nrf-arts.org/services/Scanner-WS</wsa:To>
       <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
       <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/Release</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
       <ns3:Release />
  </soapenv:Body>
</soapenv:Envelope>
```

13. Client entity service closes scanner, unregisters for events (Signed with Client private key)

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/"
standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa=”http://www.w3.org/2005/08/addressing” xmlns:ns3="http://www.nrf-
arts.org/UnifiedPOS/Scanner/">
  <soapenv:Header>
       <wsa:To>http://scanner.WS-POS-example.nrf-arts.org/services/Scanner-WS</wsa:To>
       <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
       <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/Close</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
       <ns3:Close>
            <ns3:EndpointAddress>http://192.168.1.1:8082/axis2/services/Client-
WS</ns3:EndpointAddress>
       </ns3:Close>
  </soapenv:Body>
</soapenv:Envelope>
```

14. Scanner device service notifies UDDI to delete service
15. Power off scanner device

3.3 Use Case Catalog

This section provides example usage for the WS-POS Specification. They were provided by domain experts from the OPOS-J Initiative Business Scenario Working Group Contributors in Japan. These use cases were left as is in order to preserve the original intended meaning as provided by the international community.

These are meant as examples to aid in the understanding of the WS-POS specification and are not meant as an exhaustive list for usage.

3.4 Scope

Contributor: Seiko Epson Corporation

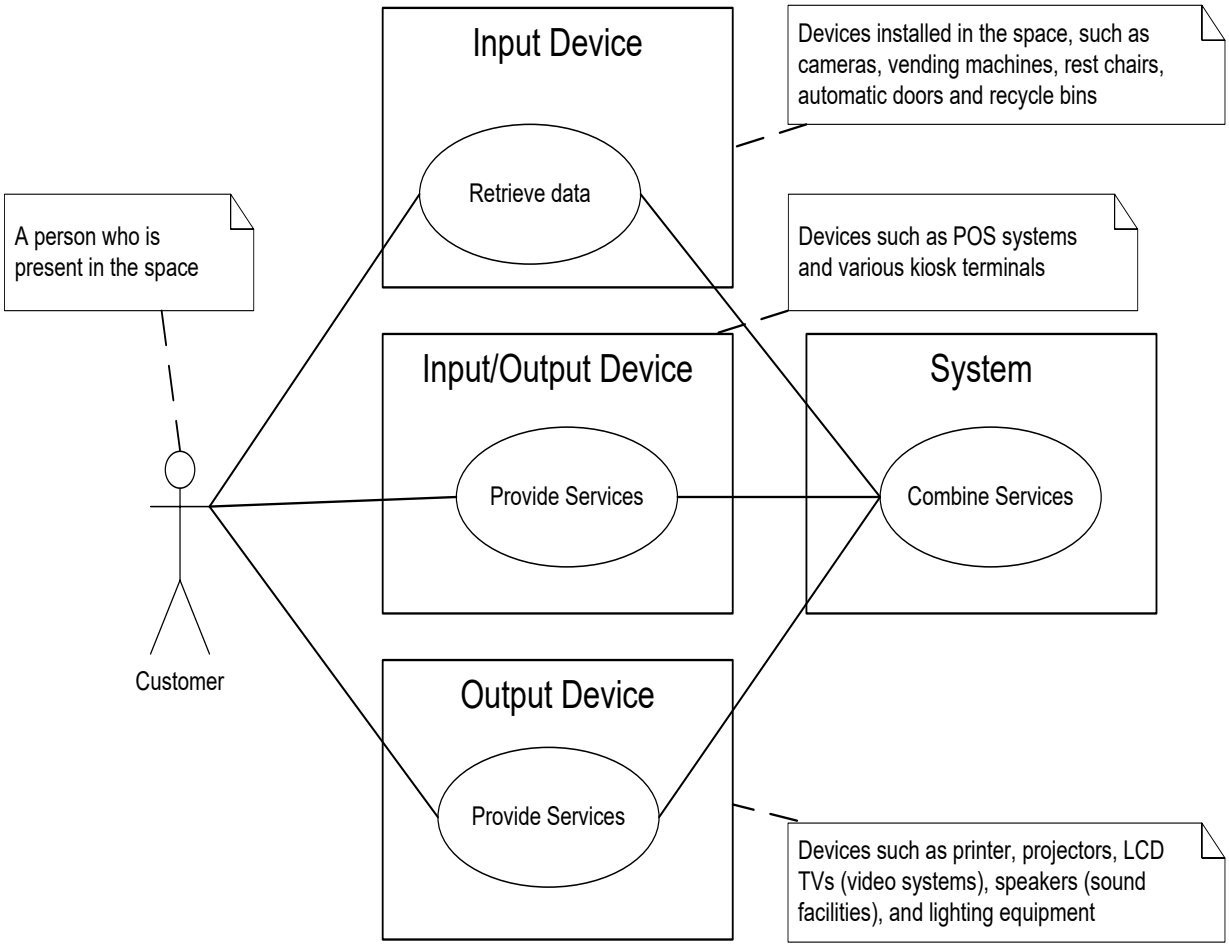
This section describes a high-level use case that covers new business scenarios based on WS-POS in general. The subsequent use cases are the embodiment or combination of the basic use case described here.

Space where all devices and applications behave as services

For an environment where input devices, output devices and input/output devices placed in every space (including rest areas, common areas, as well as every store in shopping malls) behave as services, the system offers an application program that provides new services by combining various devices to generate space where a wide variety of ideas can suggest new services to customers.

Definitions of entities:

Name	Definition
Customer	A person who is present in the space
Input device	Devices installed in the space, such as cameras, vending machines, rest chairs, automatic doors, and recycle bins
Output device	Devices such as printers, projectors, LCD TVs (video systems), speakers (sound facilities), and lighting equipment
Input/output device	Devices such as POS systems and various kiosk terminals
Application service	An application program that provides services by combining the above input, output, and input/output devices


Input Device
Retrieve data
Devices installed in the space, such as cameras, vending machines, rest chairs, automatic doors and recycle bins
A person who is present in the space
Devices such as POS systems and various kiosk terminals
Input/Output Device
Provide Services
System
Combine Services
Customer
Output Device
Provide Services
Devices such as printer, projectors, LCD TVs (video systems), speakers (sound facilities), and lighting equipment

High-Level Use Case

Use Case –	
Stakeholders	**Concerns**
Customer	A customer wants to have a comfortable time in the space. A customer also wants to enjoy satisfactions through delicate and better services.
Store manager	A store Manager wants to draw more customers and increase sales by providing high-value-added services to customers.
Device vendor	A device vendor wants to increase sales by providing easy-to-use devices and bringing the possibility of enhancing the added value of devices beyond expectation.
Software vendor	A software vendor wants to increase sales by providing the new value through flexible combination of various device functions and new suggestions to Store Managers through embodiment of various ideas.
Actors	**Description**
Customer	A person who visits the space to do some shopping or kill time.
Input device	In addition to input devices such as surveillance cameras, information of sensors attached to automatic doors may be supposed to be input devices. When sales information of vending machines can be managed in real time, the information can also be used as an input device.
Output device	In addition to output devices such as printers installed on kiosk terminals, video output devices (including projectors and LCD TVs), sound output devices (including speakers), and software-controlled lighting and air conditioning equipment can be used as output devices.
Input/output device	Devices that have both input and output device functions (such as POS terminals and kiosk terminals) can be used both as input devices and output devices.
Application program	When each input, output, input/output device can be used as service equipment, controlling output devices based on information from input devices enables programs to provide a wide variety of services depending on ideas.

Major Normal Scenario:

Label	Scenario
1.	**A customer** inputs information from **Input Devices** (but not always inputs information consciously)
1.1.	**An input Device** distributes the input information to **Application Programs**
1.2.	**An application Program** analyzes the input information and set up output information
1.3	**An application Program** sends the output information to **Output Devices**
1.4	**An output Device** distributes the output information to **Customers**
1.5	**A customer** receives the output information from **Output Devices** (but not always receives the information consciously)

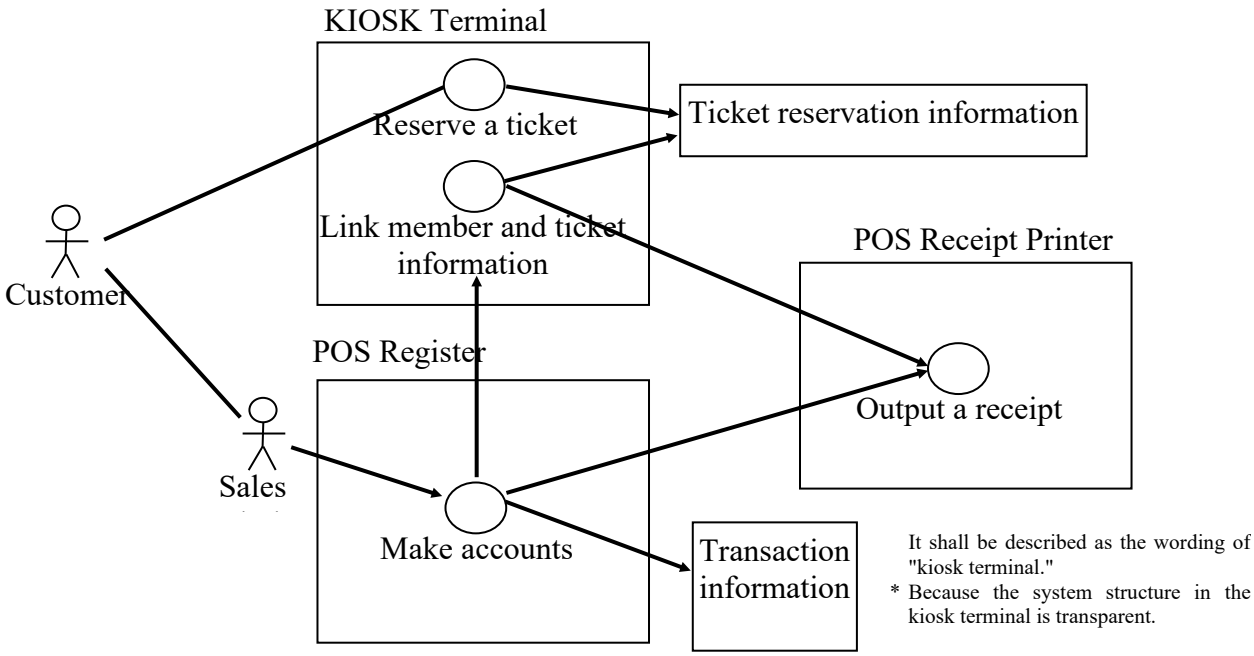
3.5 Sub Scope: Linkage between In-Store Kiosk and POS (Devices)

Contributor: Vinculum Japan Corporation

When a customer checks information (such as seat reservation information of a movie theater or an airplane) with an in-store kiosk and reserves a ticket, the system outputs the reservation information from a POS printer on the LAN.

Definitions of entities:

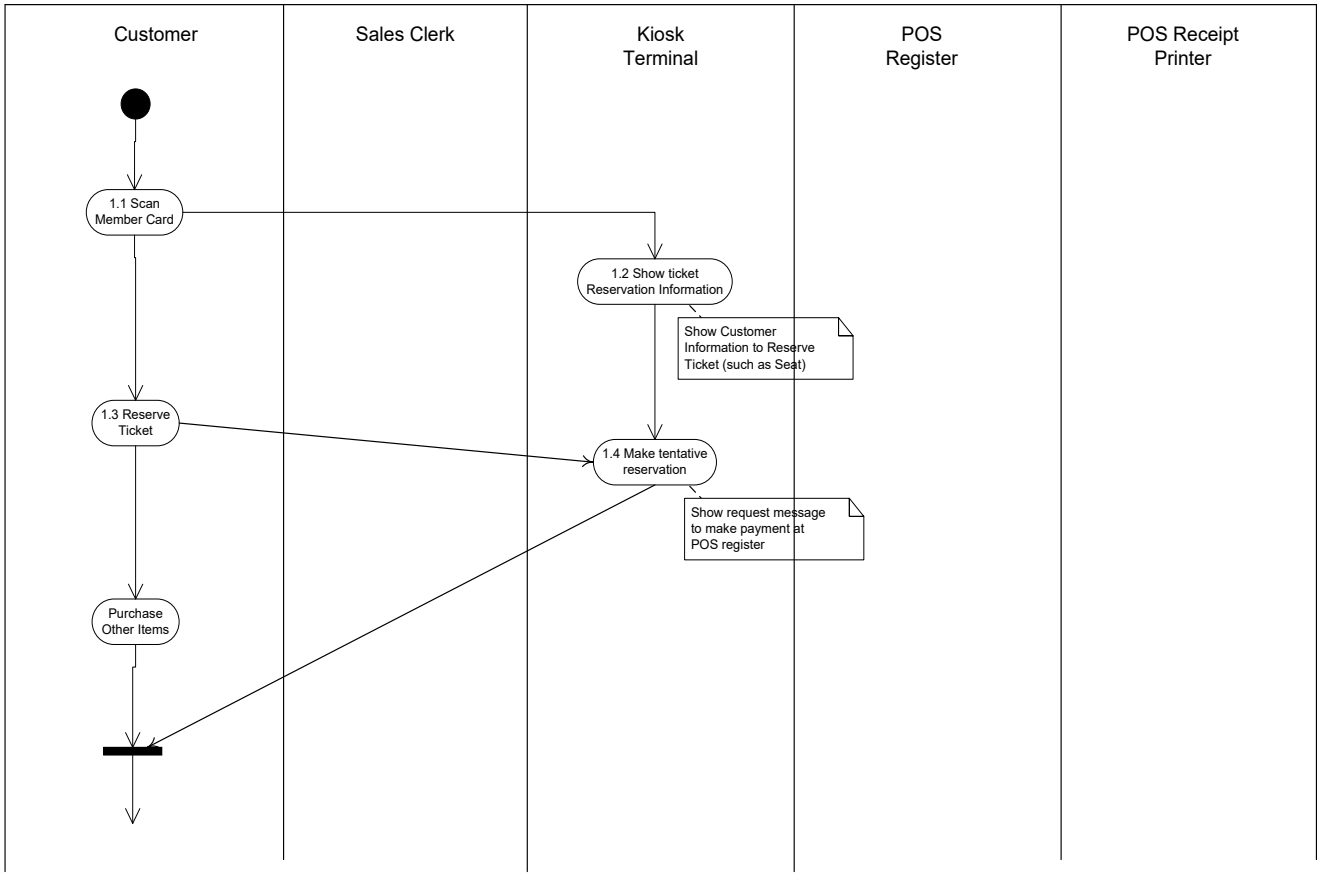
Name	Definition
Customer	A person who reserves a ticket or purchases goods.
Kiosk terminal	A terminal that manages various information display and ticket reservations.
POS register	A terminal to make accounts for in-store goods.
Sales clerk	A person who operates the POS register.
POS receipt printer	A terminal to output transaction receipts or ticket reservation receipts.
Ticket reservation information	Information (such as information on seat reservations) managed by the kiosk terminal.
Transaction information	Actual accounting information managed by the POS register.

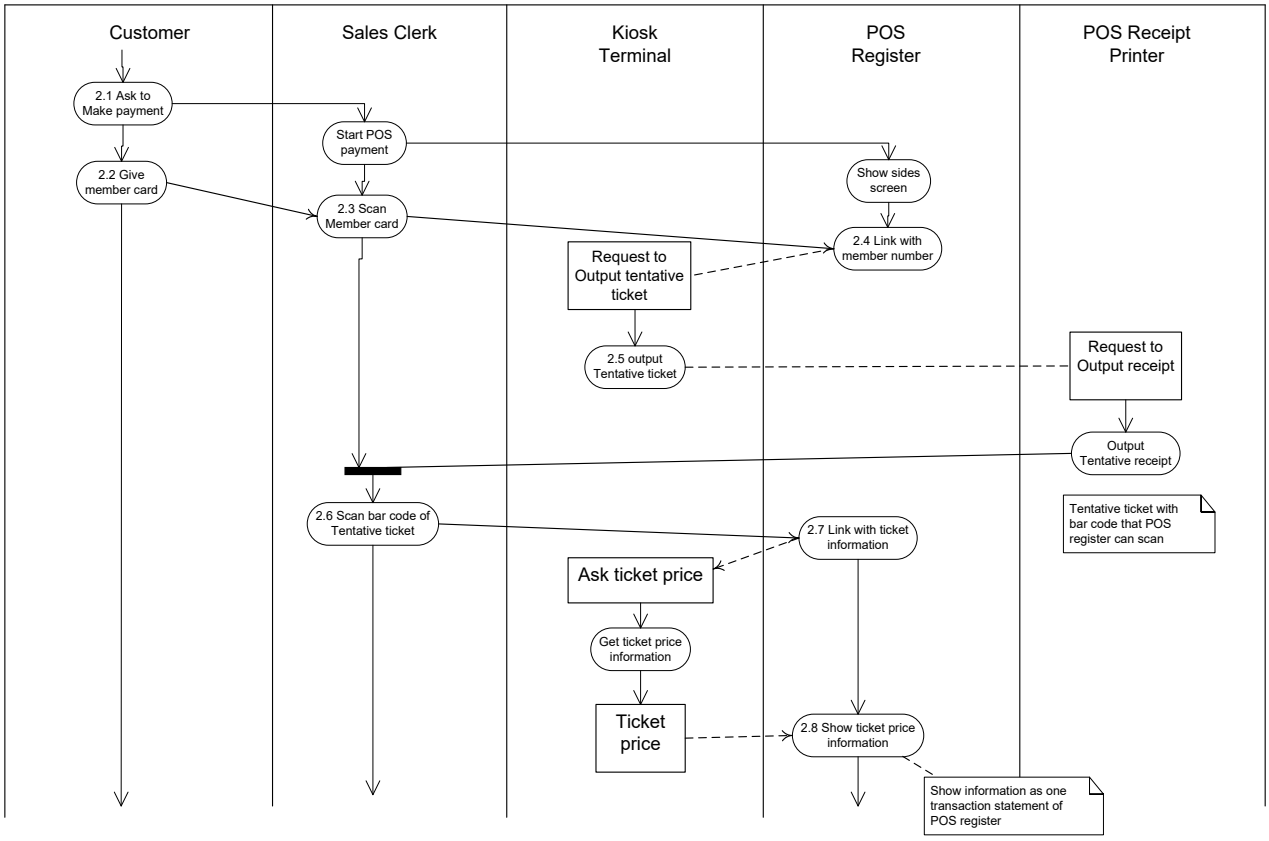


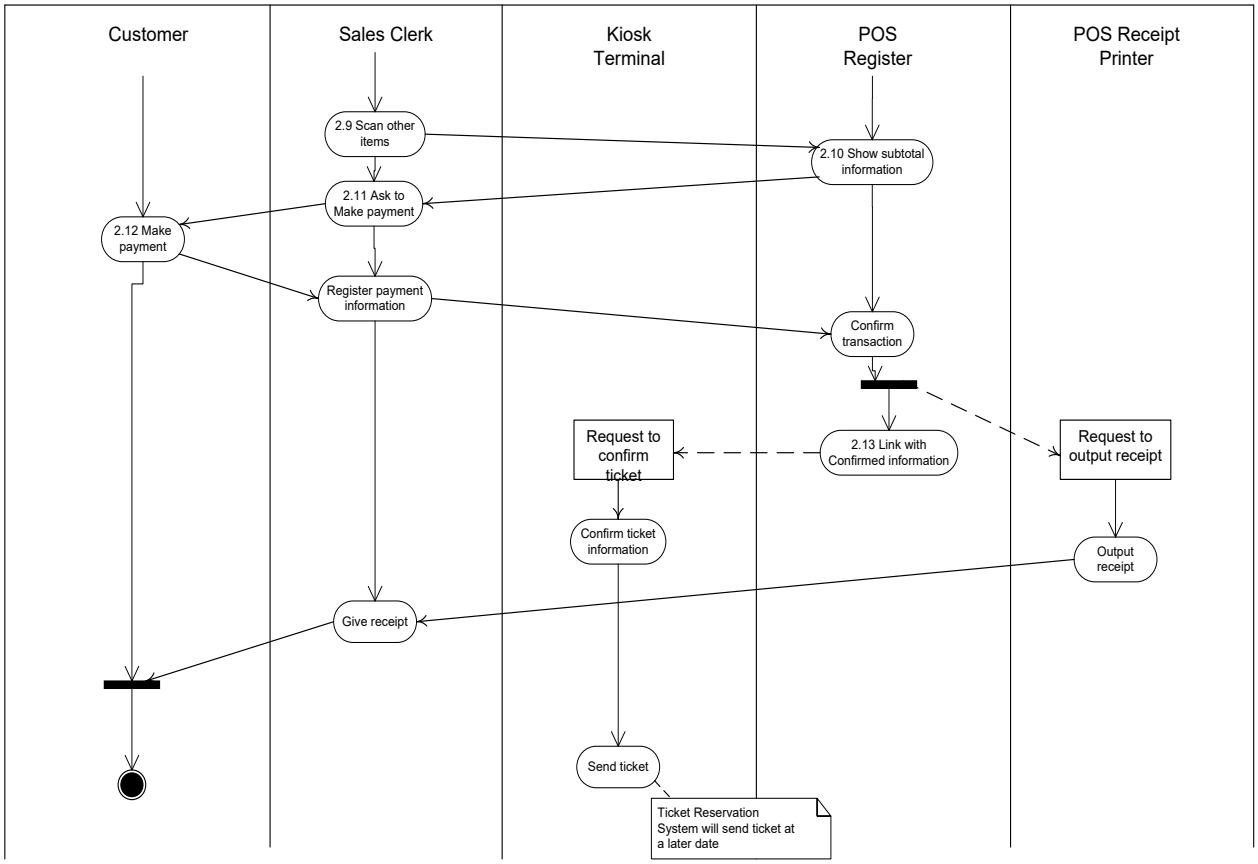
It shall be described as the wording of "kiosk terminal."

* Because the system structure in the kiosk terminal is transparent.

Activity Diagram




Customer
Sales Clerk
Kiosk Terminal
POS Register
POS Receipt Printer
2.1 Ask to Make payment
Start POS payment
2.2 Give member card
2.3 Scan Member card
Show sides screen
2.4 Link with member number
Request to Output tentative ticket
2.5 output Tentative ticket
Request to Output receipt
Output Tentative receipt
Tentative ticket with bar code that POS register can scan
2.6 Scan bar code of Tentative ticket
2.7 Link with ticket information
Ask ticket price
Get ticket price information
Ticket price
2.8 Show ticket price information
Show information as one transaction statement of POS register



3.6 Sub Scope: Linkage between Sales Assistance Terminals and POS Devices

Contributor: Toshiba Tec Corporation

At apparel retailers, the system that helps sales clerks provide the following services to customers:

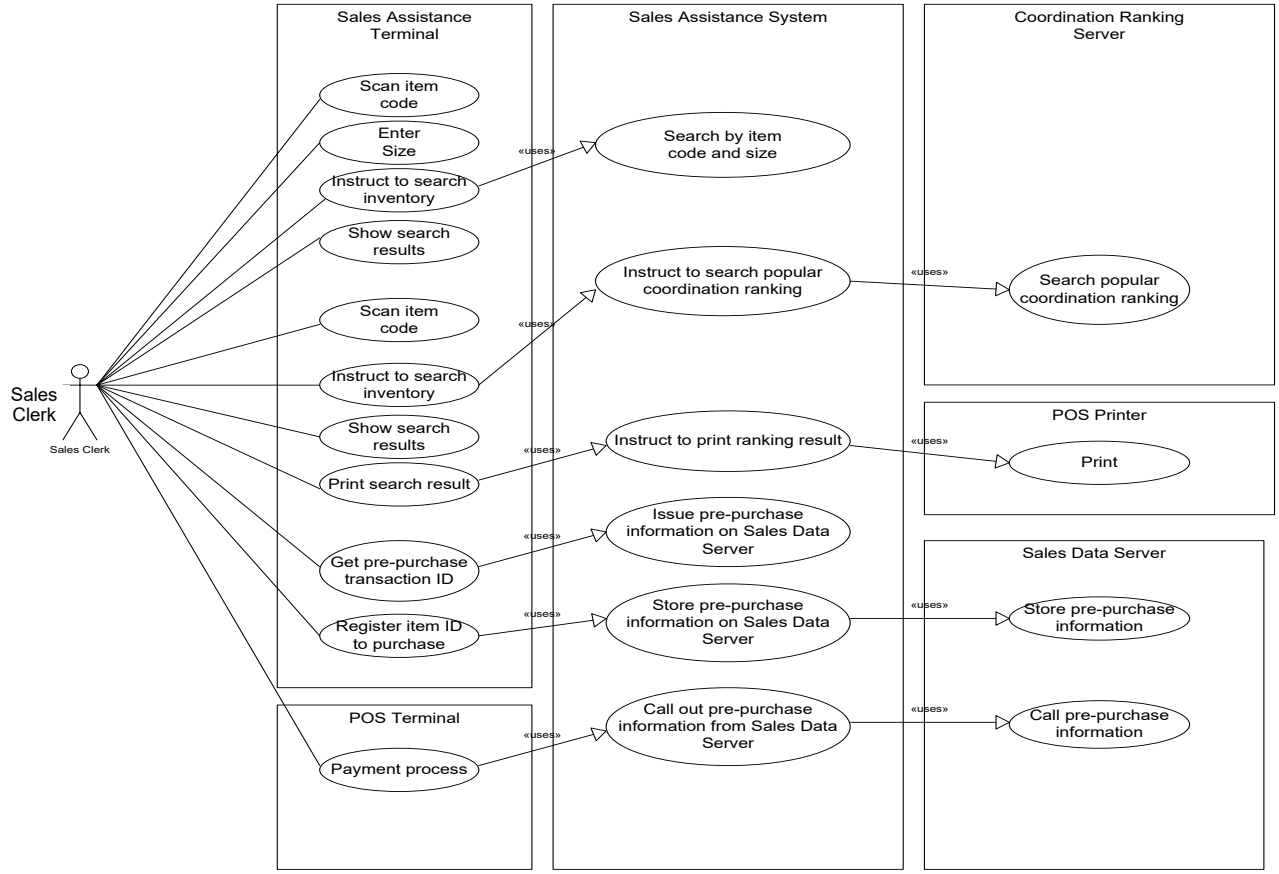
- Provide detailed information on items and the availability of items.
- Provide item-related information including arrival schedule of new items, recommended apparel coordinations, and hot-selling items.
- Pre-purchase functions before making payment with POS for customer's purchased items.

This support system consists of the following:

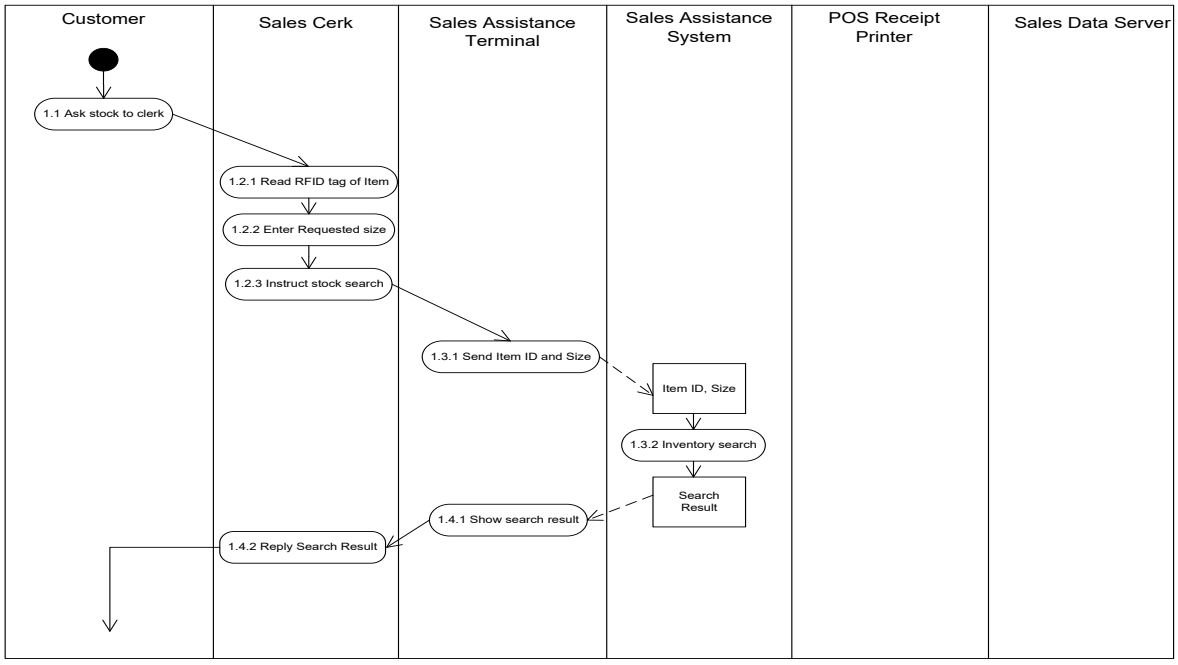
- Backend services (created with Web service applications and others) that provide search/reference functions for item information and purchase process functions,
- Sales assistance terminals carried by every sales clerk to read item codes (bar codes or RFID is used) of items and provide information to customers by collaborating with the backend services,
- POS terminals with a printer to print item information and receipts.

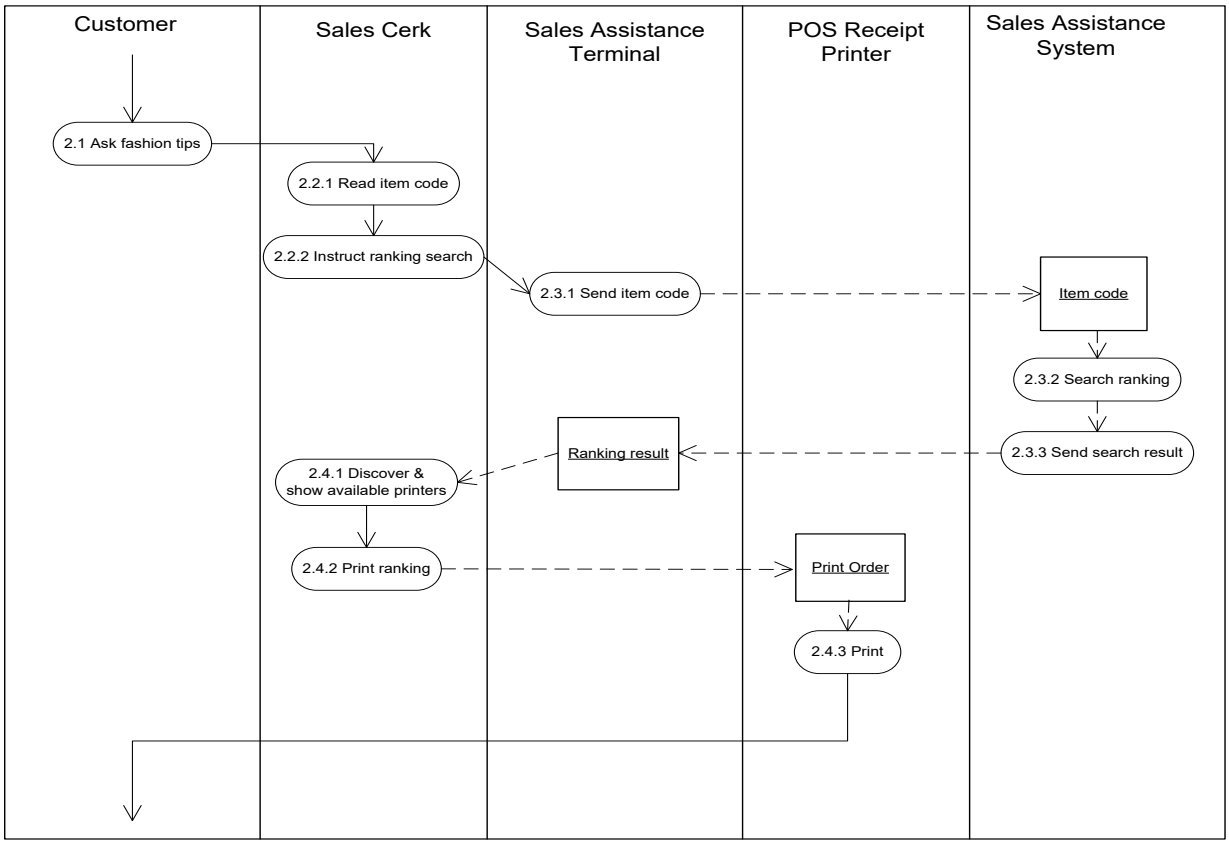
Definition of entities:

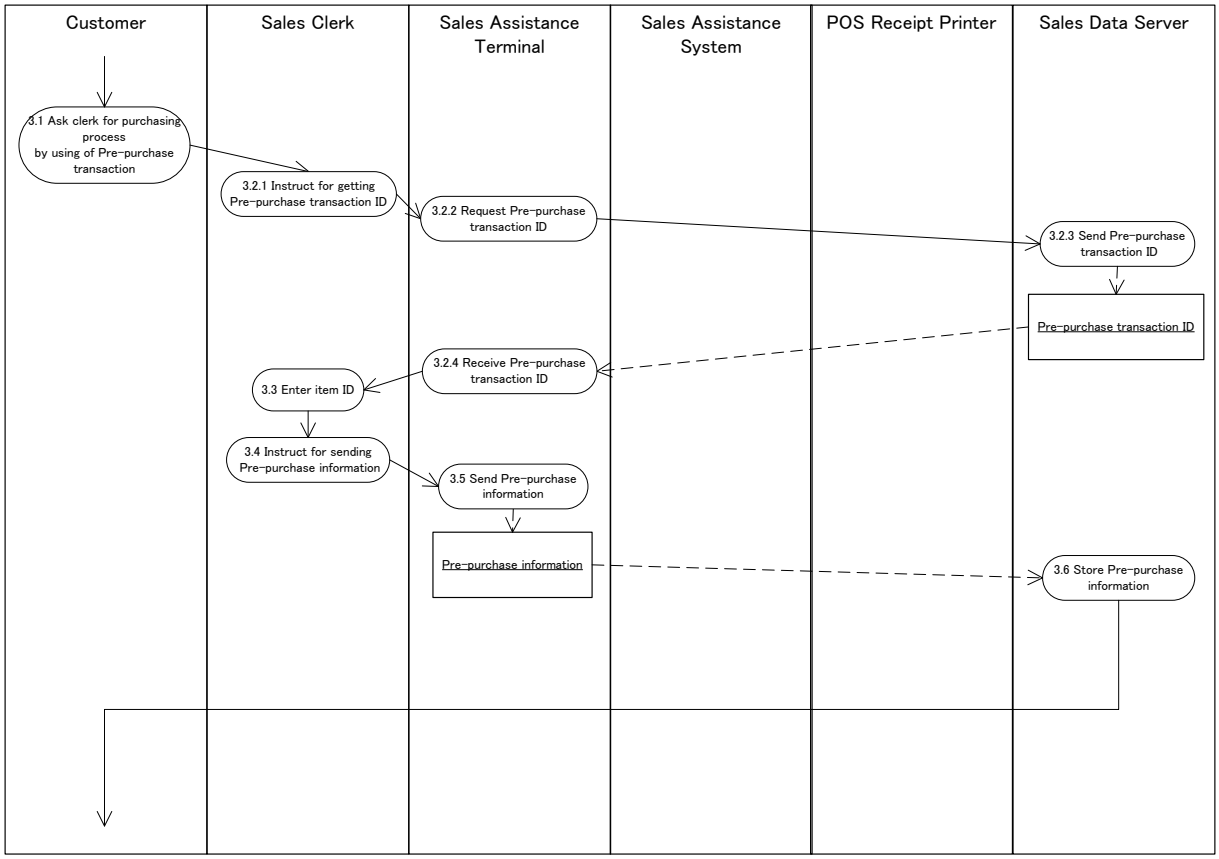
Name	Definition
Sales assistance terminal	Carried by a sales clerk, an in-store information terminal (PDA or tablet PC) for sales clerks to introduce items to customers and search inventory upon customer's request. The terminal is used as a client of the Sales Assistance System. To read item codes, the terminal is equipped with a bar code reader or an RFID reader device.
Receipt printer	A printer connected to a POS terminal to print a receipt or an acknowledgment at payment, normally.
POS terminal	A terminal used by a sales clerk when customers make purchase process. When the clerk registers items given from a customer with the terminal, the terminal displays the total amount of the purchase. Then the customer pays the amount. To handle price and discount information, general POS terminals communicate with servers that contain information on items and prices. Most POS terminals are equipped with a receipt printer, a bar code scanner, price display screens (including a screen for clerks and a screen for customers), and POS keyboard.
Item	Goods or services that a customer purchases. A customer makes its payment by cash or other means in exchange for the item to purchase. When seen from the system side, an item has secondary information including ID, name, and price.
Inventory	Items on the store shelves or stored in backyards or warehouses.
Item-related information such as coordination rankings	Reference information for customers when they purchase an item. Ranking information indicates the latest trend or popularity of coordinations. Customers refer to the information when purchasing items.

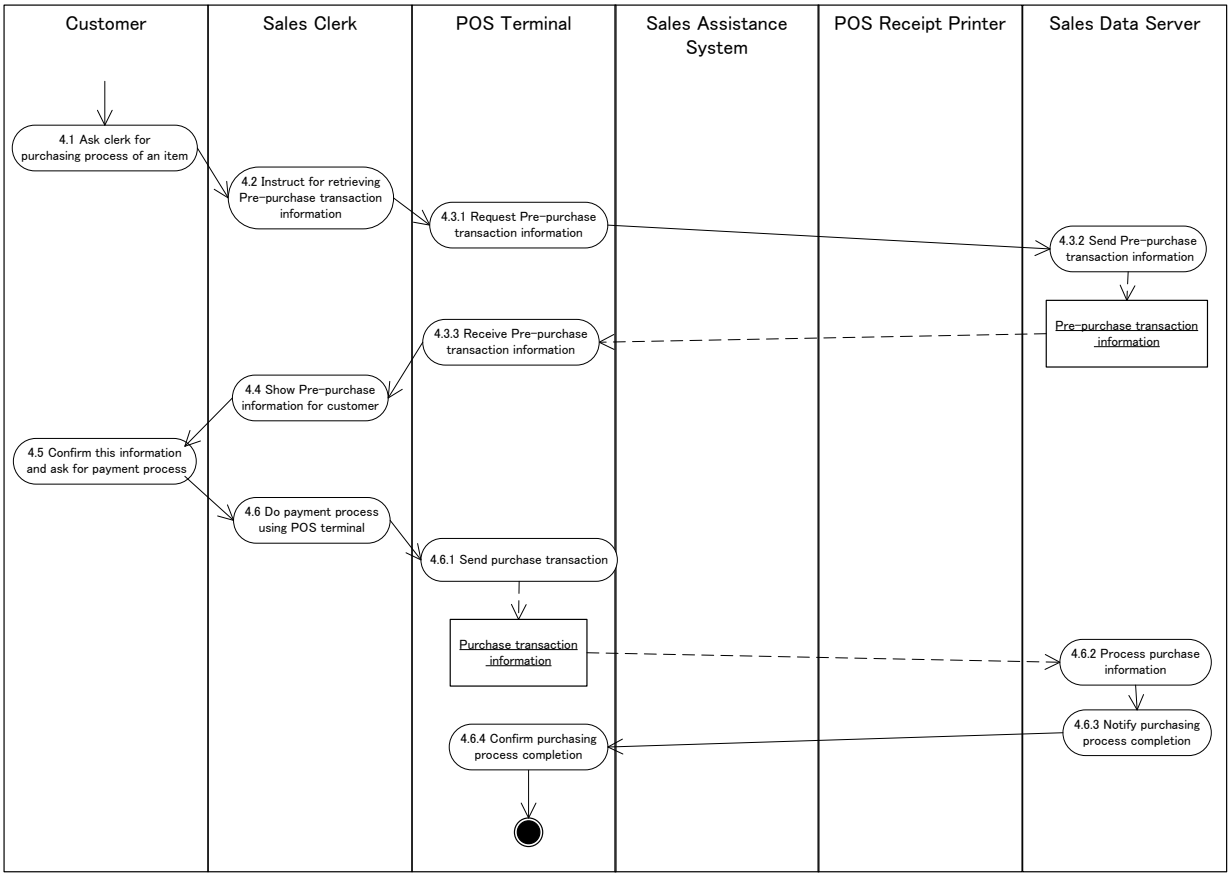


Activity Diagram




Customer
Sales Cerk
Sales Assistance Terminal
POS Receipt Printer
Sales Assistance System
2.1 Ask fashion tips
2.2.1 Read item code
2.2.2 Instruct ranking search
2.3.1 Send item code
Item code
2.3.2 Search ranking
2.3.3 Send search result
Ranking result
2.4.1 Discover & show available printers
2.4.2 Print ranking
Print Order
2.4.3 Print


Customer
Sales Clerk
Sales Assistance Terminal
Sales Assistance System
POS Receipt Printer
Sales Data Server
3.1 Ask clerk for purchasing process by using of Pre-purchase transaction
3.2.1 Instruct for getting Pre-purchase transaction ID
3.2.2 Request Pre-purchase transaction ID
3.2.3 Send Pre-purchase transaction ID
Pre-purchase transaction ID
3.2.4 Receive Pre-purchase transaction ID
3.3 Enter item ID
3.4 Instruct for sending Pre-purchase information
3.5 Send Pre-purchase information
Pre-purchase information
3.6 Store Pre-purchase information



3.7 Sub Scope: Batch Payment in Commercial Complex

Contributor: Sorimachi Giken Co., Ltd.

At a commercial complex such as a department store or a shopping mall, the system makes multiple payments at once at the dedicated payment corner or POS in any retail store instead of making individual payment at the respective store.

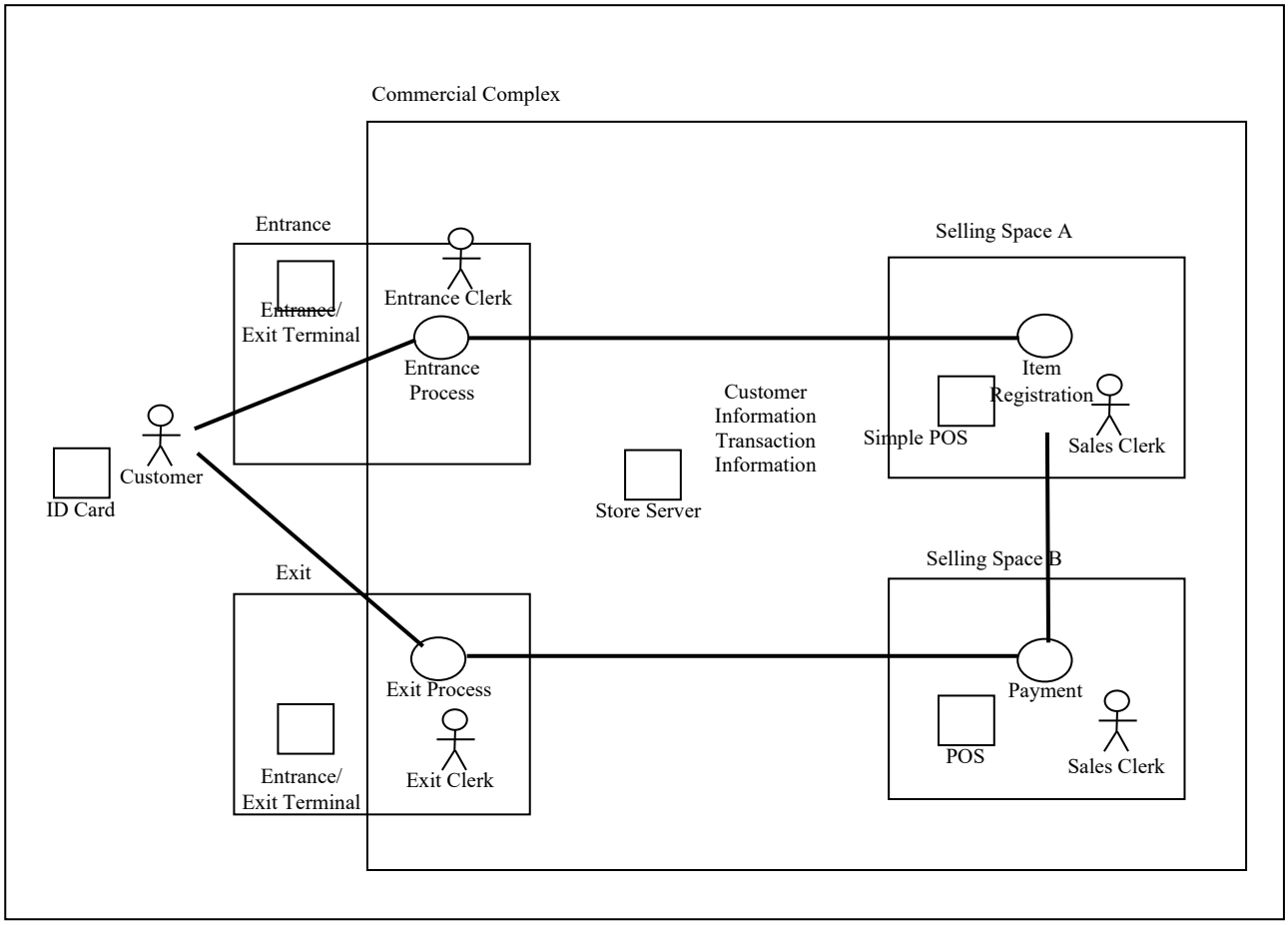
This sub scope is independent of whether a customer takes items home or makes them delivered to home.

When delivering, a customer has to write down its address only once at any store. Then the system delivers all of the customer's items purchased in the commercial complex at once.

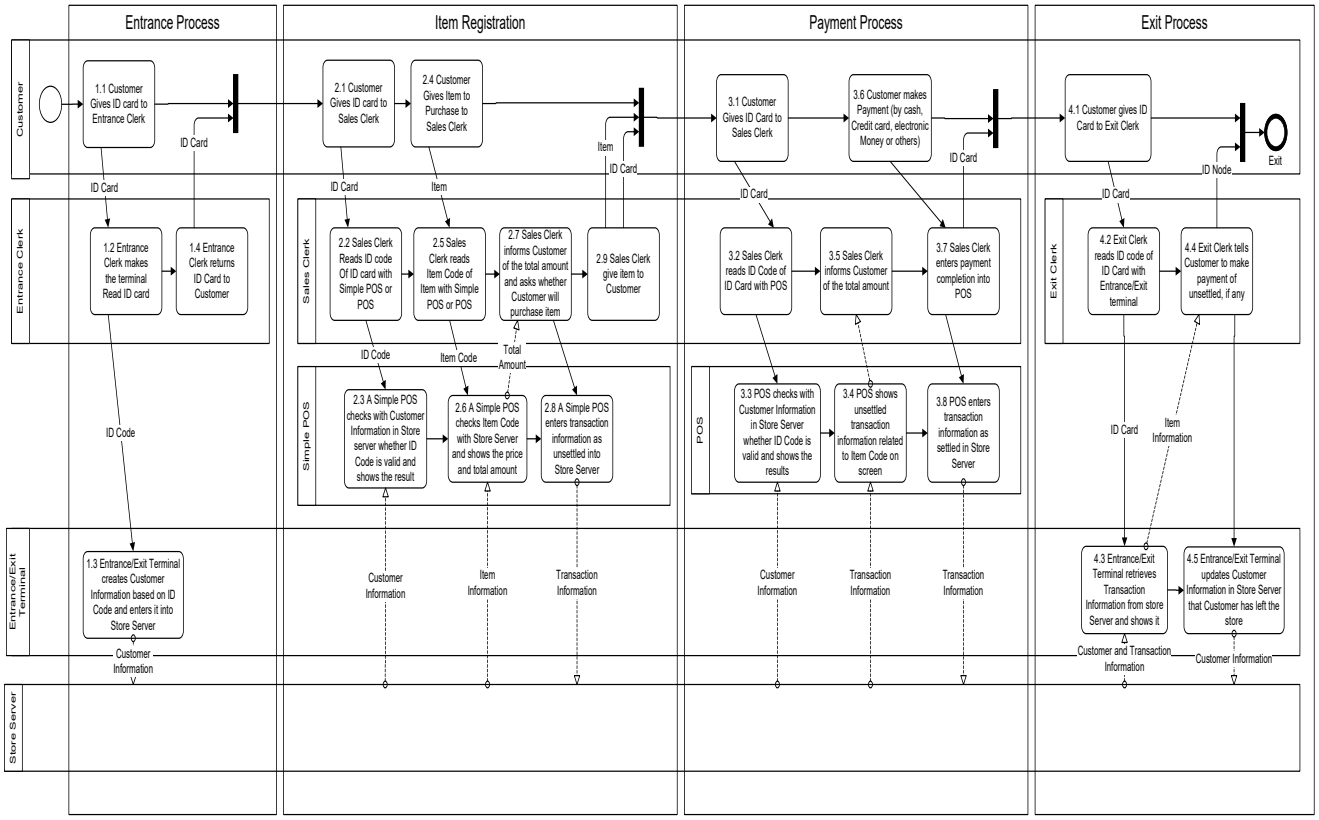
Definitions of entities:

Type	Entitiy	Description
Customer	Customer	
Clerk	Entrance Clerk	A clerk who performs an entrance process at an entrance of a commercial complex

	Exit Clerk	A clerk who performs an exit process at an exit of a commercial complex
	Sales clerk	A clerk who performs item registration process and payment process in a store in a commercial complex
Terminal	ID card	A card that identifies a customer Including member cards, credit cards, and instant issue cards
	Entrance/exit terminal	A terminal that reads or issues ID cards at an entrance or an exit A hand-held terminal equipped with a card reader for reading ID cards is assumed
	Simple POS	A POS that can only perform an item registration process A hand-held terminal equipped with a card reader for reading ID cards and a bar code scanner for reading item bar codes is assumed.
	POS	A POS that can perform an item registration process and a payment process A stationary POS equipped with a card reader for reading ID cards and a bar code scanner for reading item bar codes that can perform payment process is assumed
	Store server	A server that centrally manages data of simple POS terminals and POS terminals in a store
Data	Customer information	Information on ID cards, ID codes, and customers
	Transaction information	Information that consists of ID codes, item codes and the payment status


Commercial Complex
Entrance
Entrance/
Exit Terminal
Entrance Clerk
Entrance
Process
Selling Space A
Item
Registration
Simple POS
Sales Clerk
Customer
Information
Transaction
Information
Customer
ID Card
Store Server
Exit
Selling Space B
Exit Process
Payment
POS
Sales Clerk
Entrance/
Exit Terminal
Exit Clerk

Activity Diagram



3.8 Sub Scope: Monitoring In-Store KIOSK Equipment and Cooperation with Back-office on Occurrence of Problems.

Contributor: Fujitsu Frontech Limited

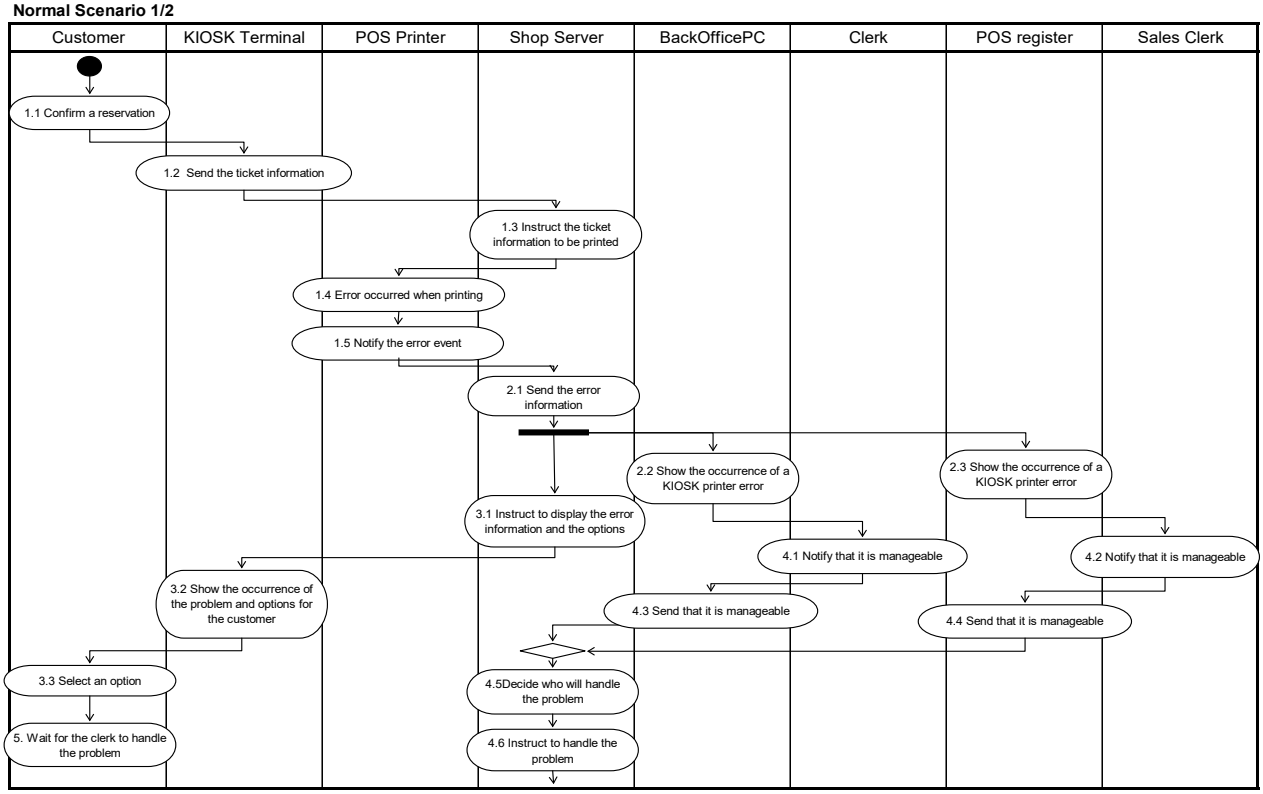
The way of handling problems which might occur when a customer tries to let the machine read a card to make payment or to print out the result with POS printer after viewing various information (such as seat-reservation information for a movie theater or an airplane) on an in-store kiosk and reserving a ticket.

Definition of entities:

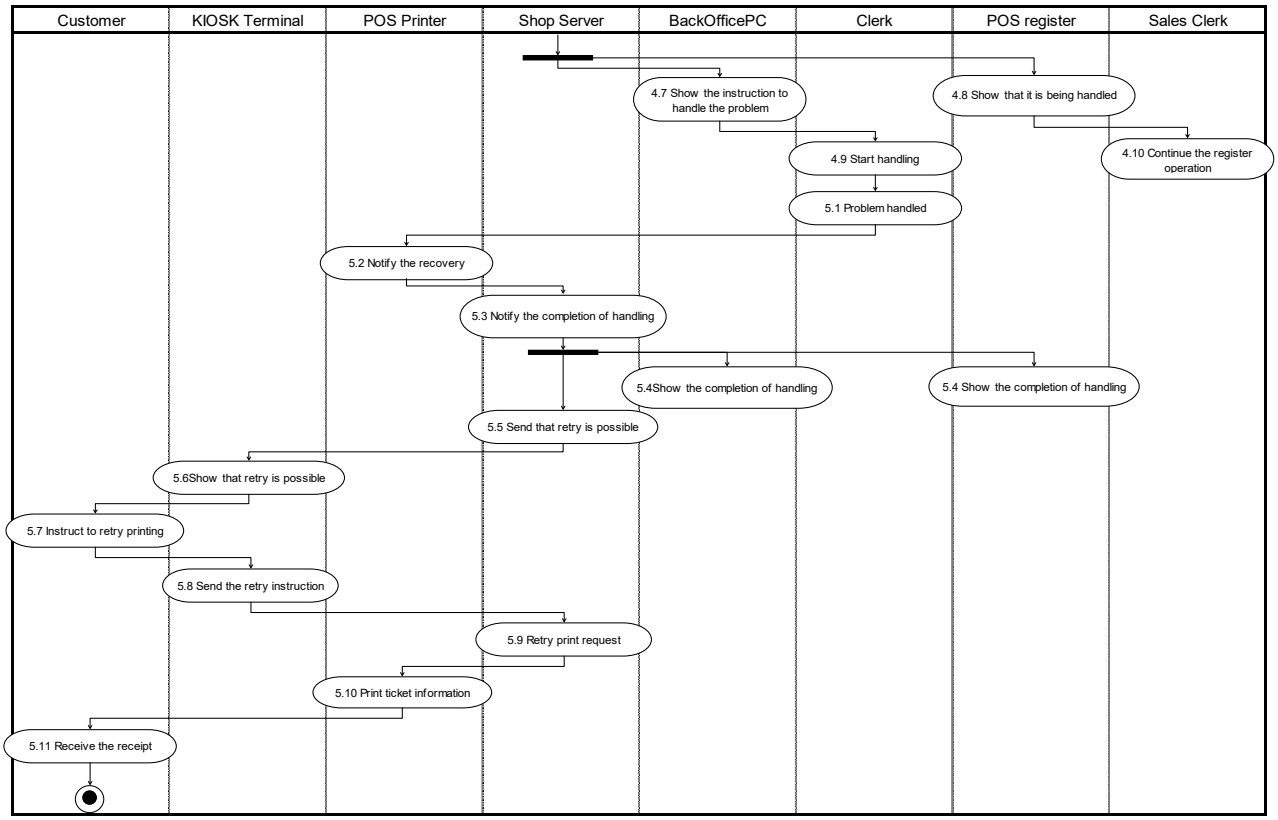
Name	Definition
Customer	Customer is someone who reserves a ticket or purchases a product.
Kiosk terminal 1	Kiosk terminal is a terminal which controls the display of various kinds of information and the reservation of tickets. Multiple terminals are installed side-by-side and making up a department
Kiosk terminal 2	
Kiosk terminal 3	
POS printer	POS printer is a component of the kiosk terminal which outputs a receipt for a transaction or a receipt for ticket reservation.
Ticket-reservation information	Ticket-reservation information is management information for seat-reservation information or others which are controlled by the kiosk terminal
Transaction information	Transaction information is information of actual results in accounting which is controlled by the POS register.
Shop server	Shop server is a server which controls services on the kiosk terminal.
Back-office PC	Back-office PC is a terminal which performs various tasks such as clerical works at the backyard of a store
POS register	POS register is a terminal used to make accounts for commodities
Sales clerk	Sales clerk is someone who operates the POS register
Clerk	Clerk is someone who performs various tasks except the operation of POS register
Electronic value R/W	Electronic value R/W is a component of the kiosk terminal which performs inputting and outputting of electronic cash and tickets.

Activity Diagram

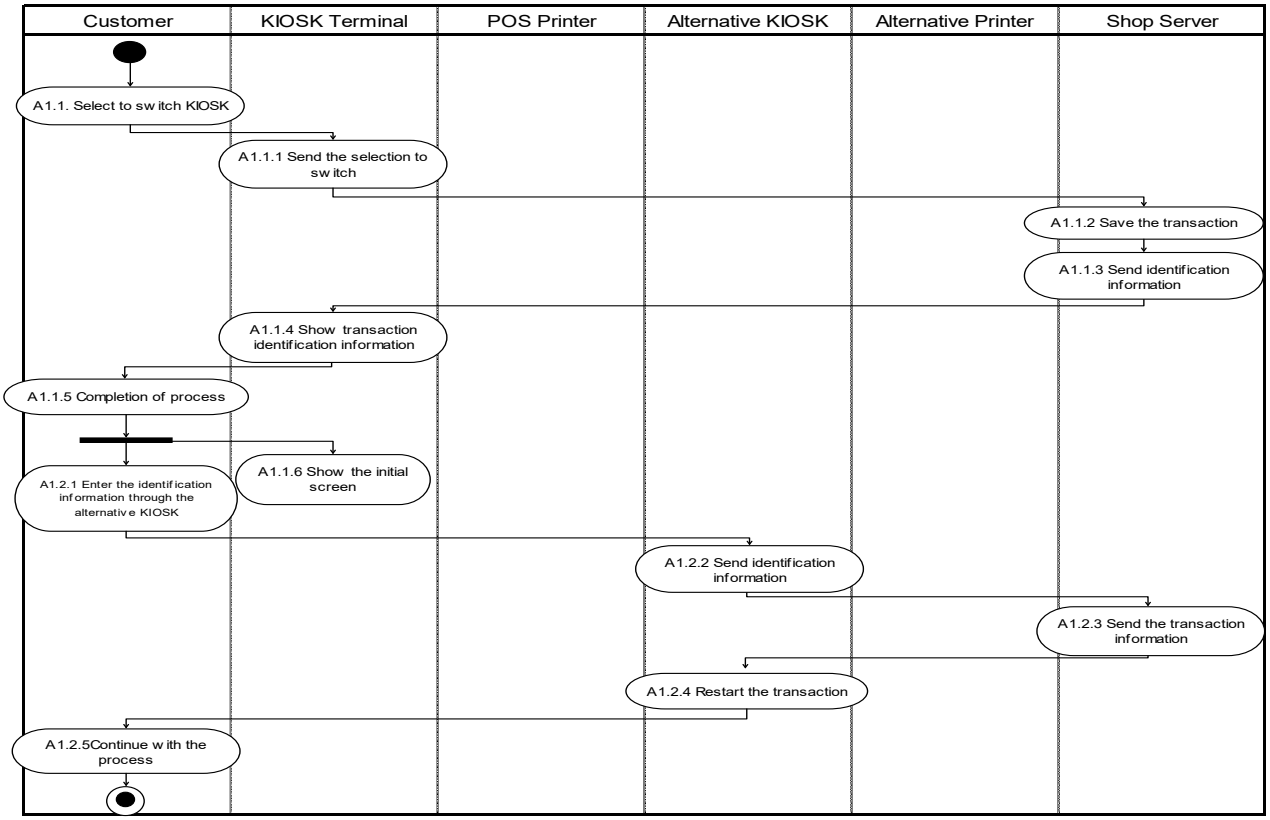
Normal Scenario 1/2



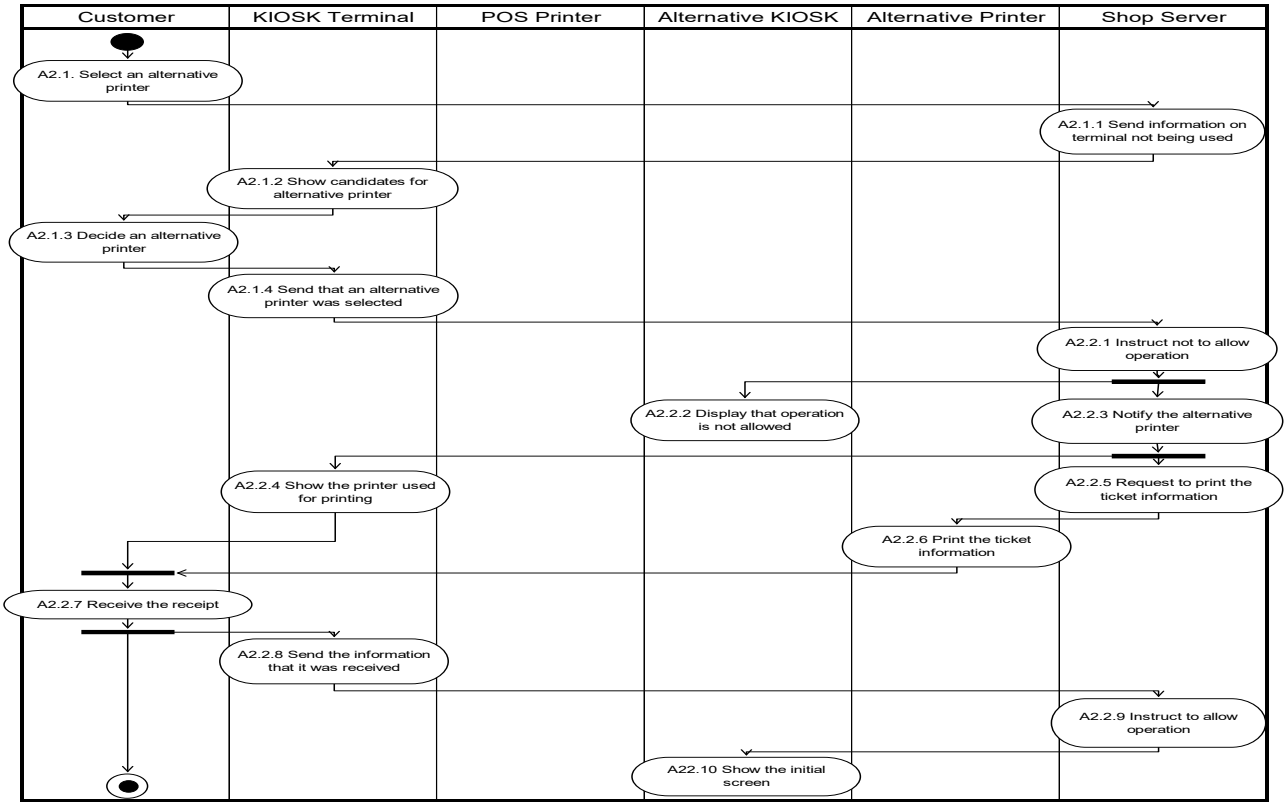
Normal Scenario 2/2



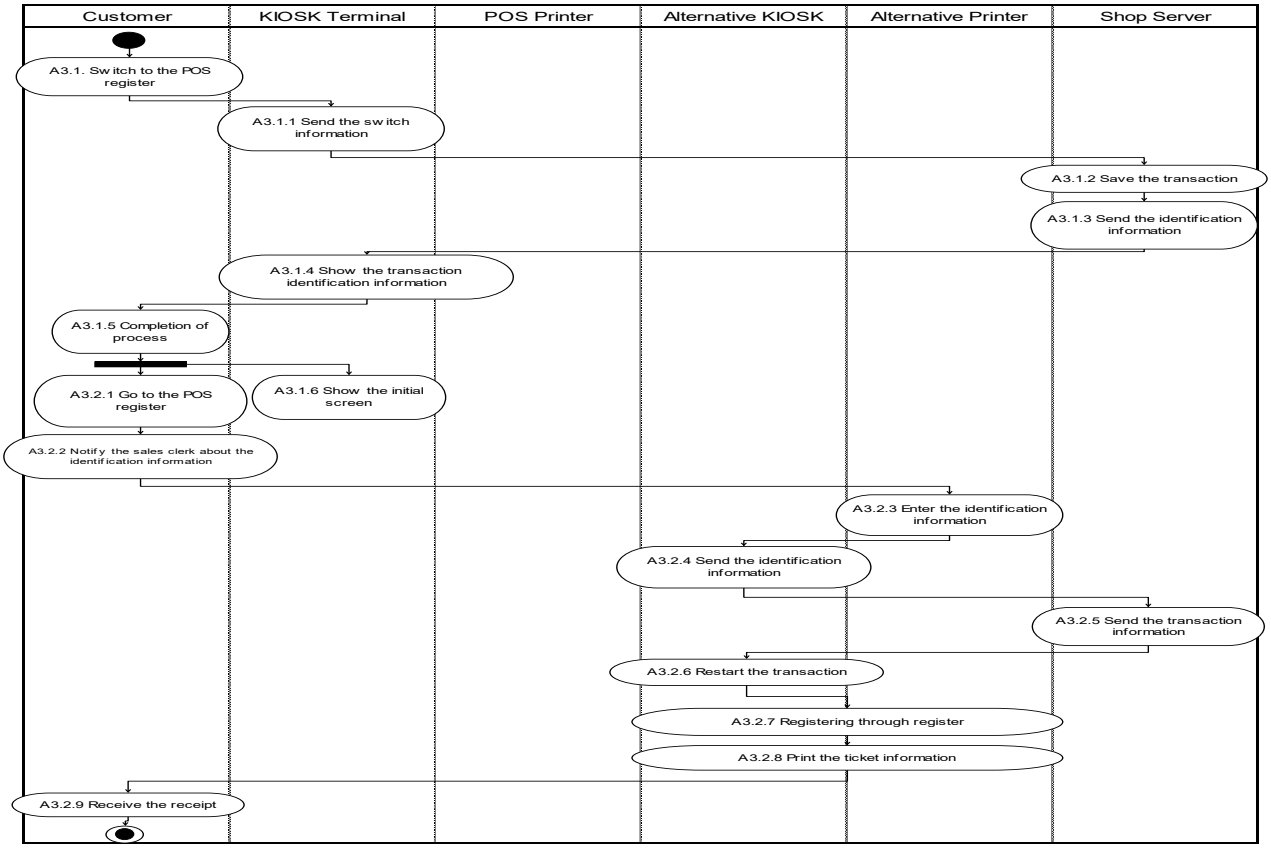
Alternative Scenario 1



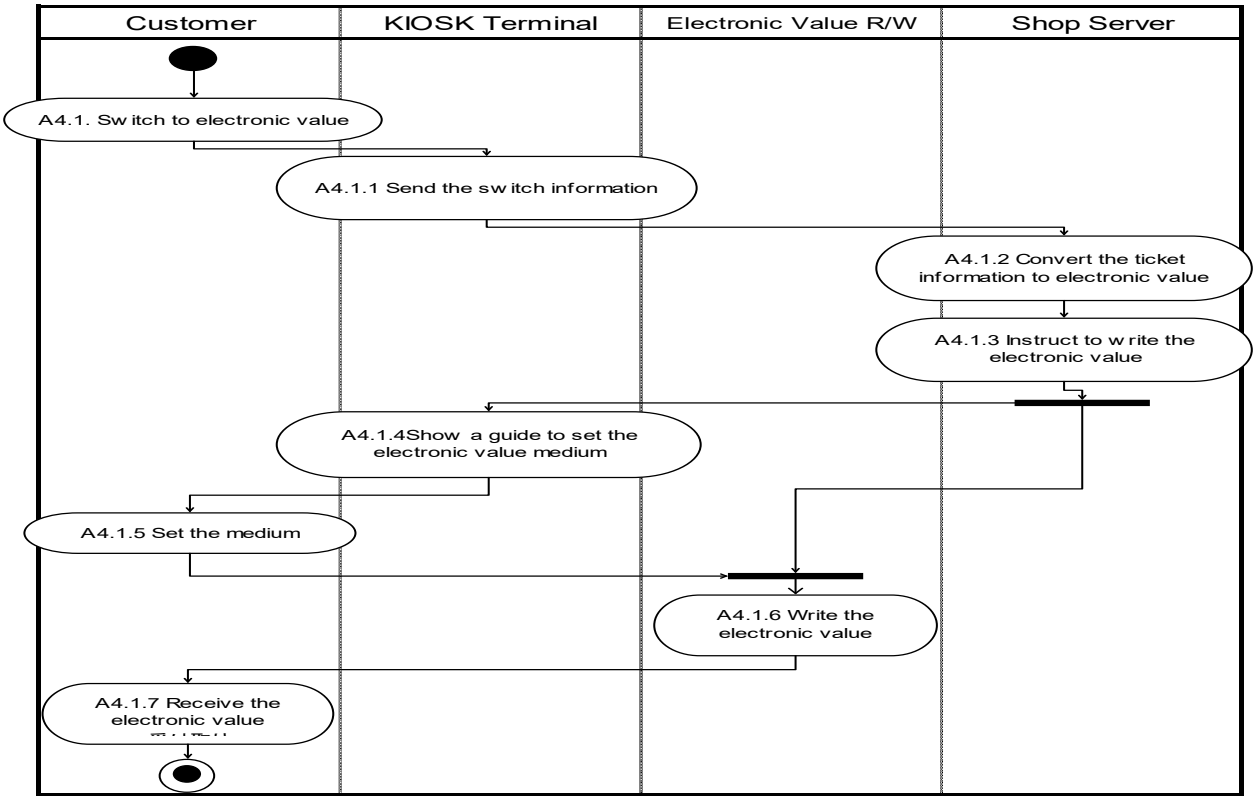
Alternative Scenario 2



Alternative Scenario 3



Alternative Scenario 4



3.9 Sub Scope: POS System in Consideration of Cooperation between Various Industries

Contributor: Star Micronics Co., Ltd.

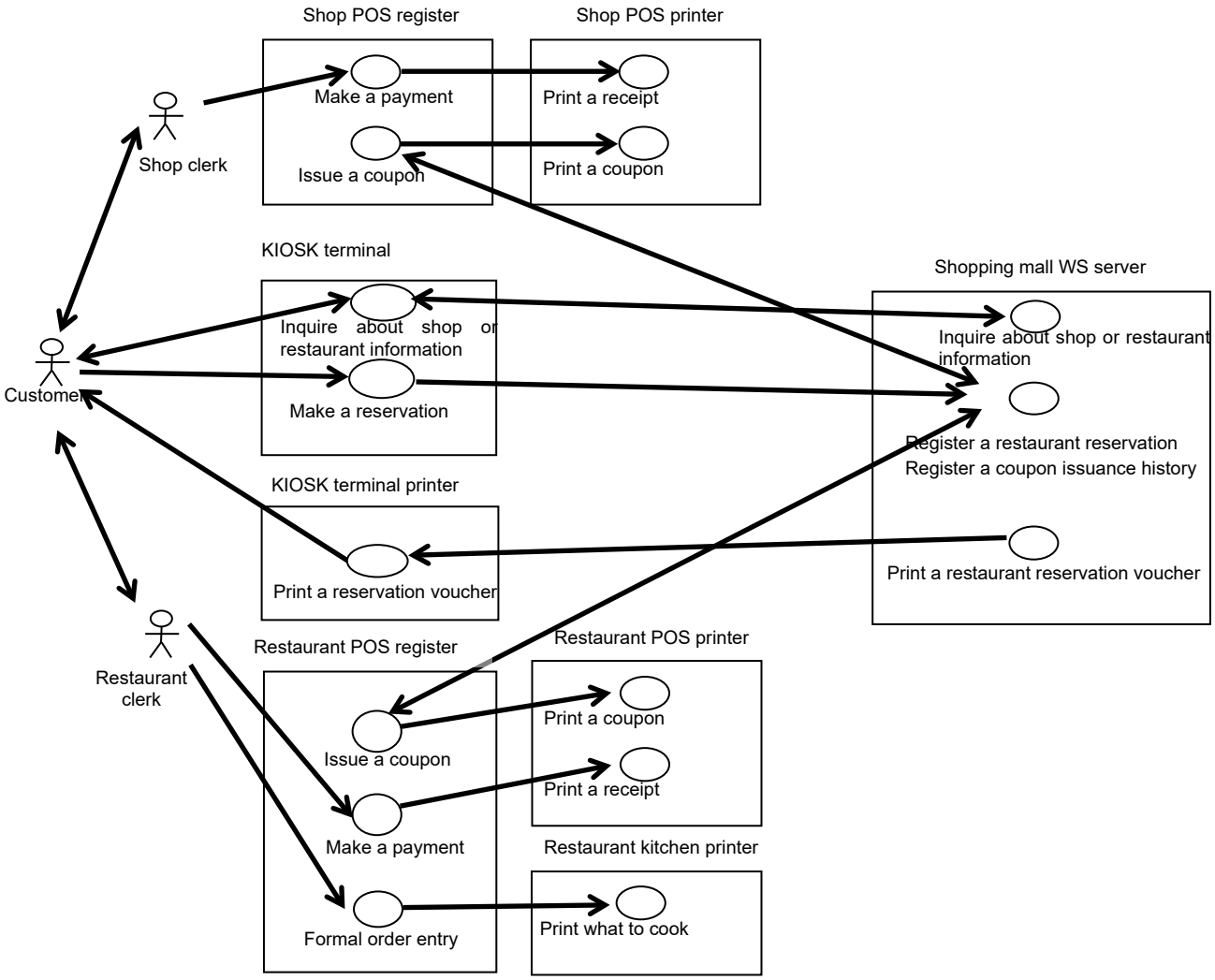
A customer purchases goods at a retail store or has a meal at a restaurant.

When making payment at a register, a coupon is issued which can be used at another industry (such as at a restaurant) along with issuing a receipt.

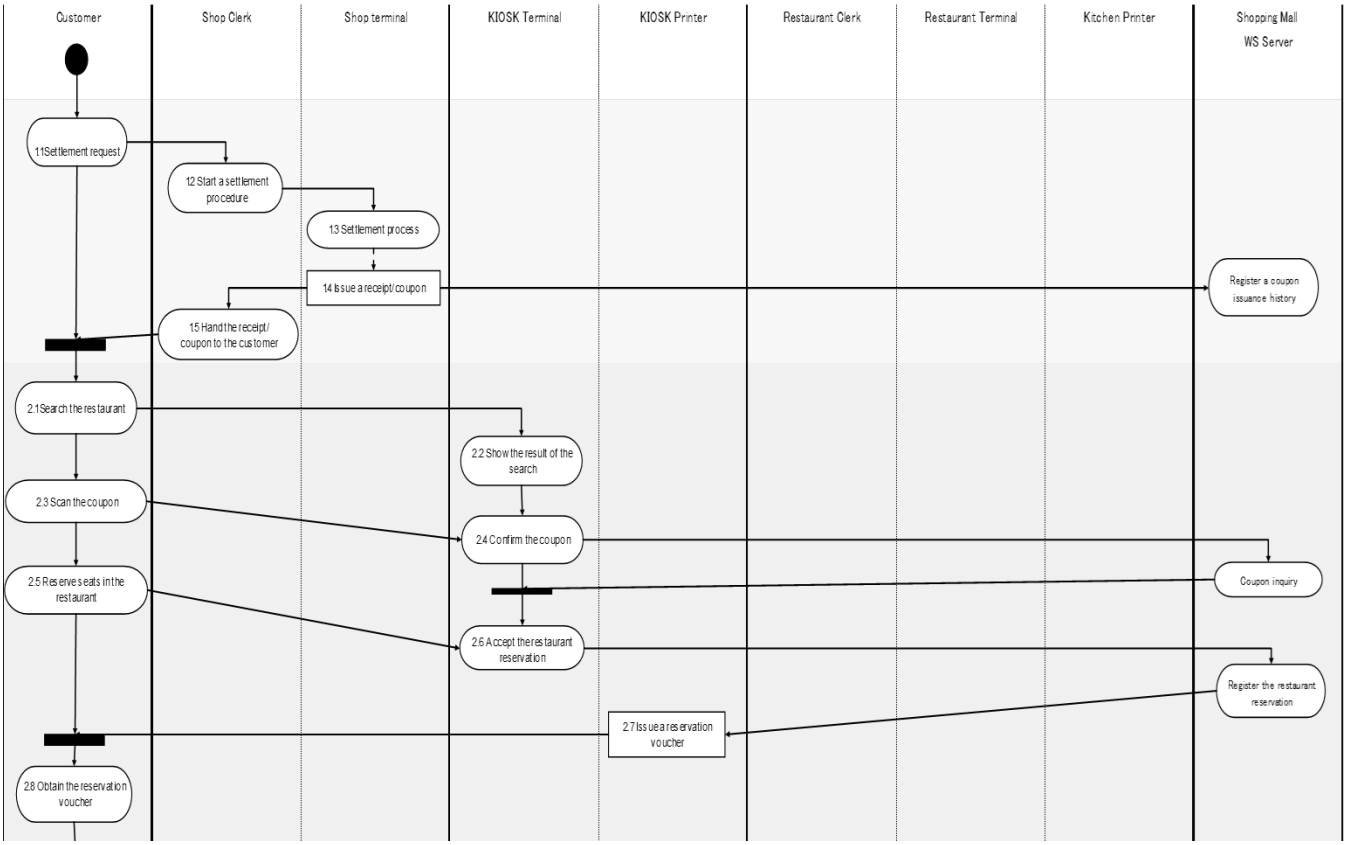
Reservation of seats at a restaurant or the like is possible with a kiosk installed within a shopping mall using this coupon.

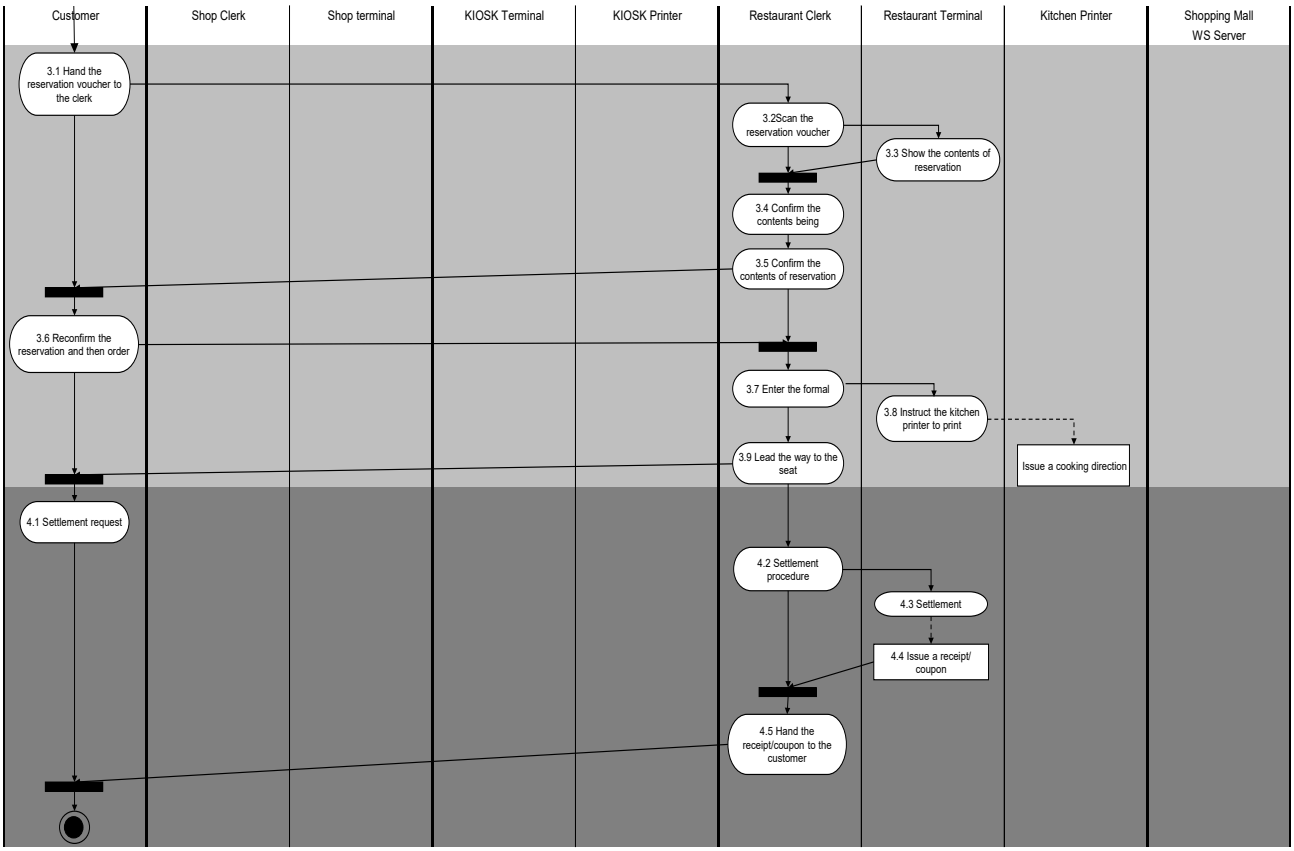
Definition of entities:

Name	Definition
Customer	Someone who purchases goods or has a meal at a restaurant.
Shop clerk	Someone who operates the POS register in the shop
Shop kiosk terminal	A terminal which retrieves and displays bargain information for shopping in the mall or which is used to reserve seats at a restaurant. This terminal is operated by the customer.
Shop information system	An information system which controls the receiving and placing of orders and performs customer management for the shop
Shop POS terminal	A POS terminal installed in a shop. This terminal is operated by a clerk at the shop
Restaurant clerk	Someone who arranges seats for customer and who receives order for dishes at a restaurant
Restaurant information system	An information system which controls the receiving and placing of orders and performs seat management and other tasks in the restaurant.
Restaurant terminal	A terminal for the information system which controls the receiving and placing of orders and performs seat management and other tasks in the restaurant. This terminal is operated by a clerk at the restaurant
Restaurant kitchen printer	A printer installed in the kitchen of a restaurant. It prints what to cook at the appropriate time
Web Base Server System	A POS mutual mediation system shared within a shopping mall to support web-based data communication.


Shop POS register
Shop POS printer
Make a payment
Print a receipt
Issue a coupon
Print a coupon
Shop clerk
KIOSK terminal
Shopping mall WS server
Inquire about shop or restaurant information
Inquire about shop or restaurant information
Make a reservation
Register a restaurant reservation
Register a coupon issuance history
Customer
KIOSK terminal printer
Print a reservation voucher
Print a restaurant reservation voucher
Restaurant clerk
Restaurant POS register
Restaurant POS printer
Print a coupon
Issue a coupon
Print a receipt
Make a payment
Restaurant kitchen printer
Formal order entry
Print what to cook

Activity Diagram




Customer
Shop Clerk
Shop terminal
KIOSK Terminal
KIOSK Printer
Restaurant Clerk
Restaurant Terminal
Kitchen Printer
Shopping Mall WS Server
3.1 Hand the reservation voucher to the clerk
3.2Scan the reservation voucher
3.3 Show the contents of reservation
3.4 Confirm the contents being
3.5 Confirm the contents of reservation
3.6 Reconfirm the reservation and then order
3.7 Enter the formal
3.8 Instruct the kitchen printer to print
Issue a cooking direction
3.9 Lead the way to the seat
4.1 Settlement request
4.2 Settlement procedure
4.3 Settlement
4.4 Issue a receipt/ coupon
4.5 Hand the receipt/coupon to the customer

3.10 Sub Scope: Linkage between Display Shelf and Rear System

Contributor: OMRON SOFTWARE Co., Ltd.

Definition of entities (Part 1):

Name	Definition
Shelf replacing worker	A shelf replacing worker is someone who places relevant commodities on shelves while performing shelving allocation
Electronic inventory tag with scanner	An electronic inventory tag with scanner is a device which reads bar code indicating the product name and the location of the shelf and displays the commodity information such as commodity price (However, this device does not exist at present)
Store controller	A server which controls commodity information (such as product name, selling price, quantity, and others).
Shelving allocation system	A server which controls the state of shelving allocation on display shelf
Electronic inventory tag system (Rear)	A server which instructs the electronic inventory tag with scanner to display information and controls the state of display

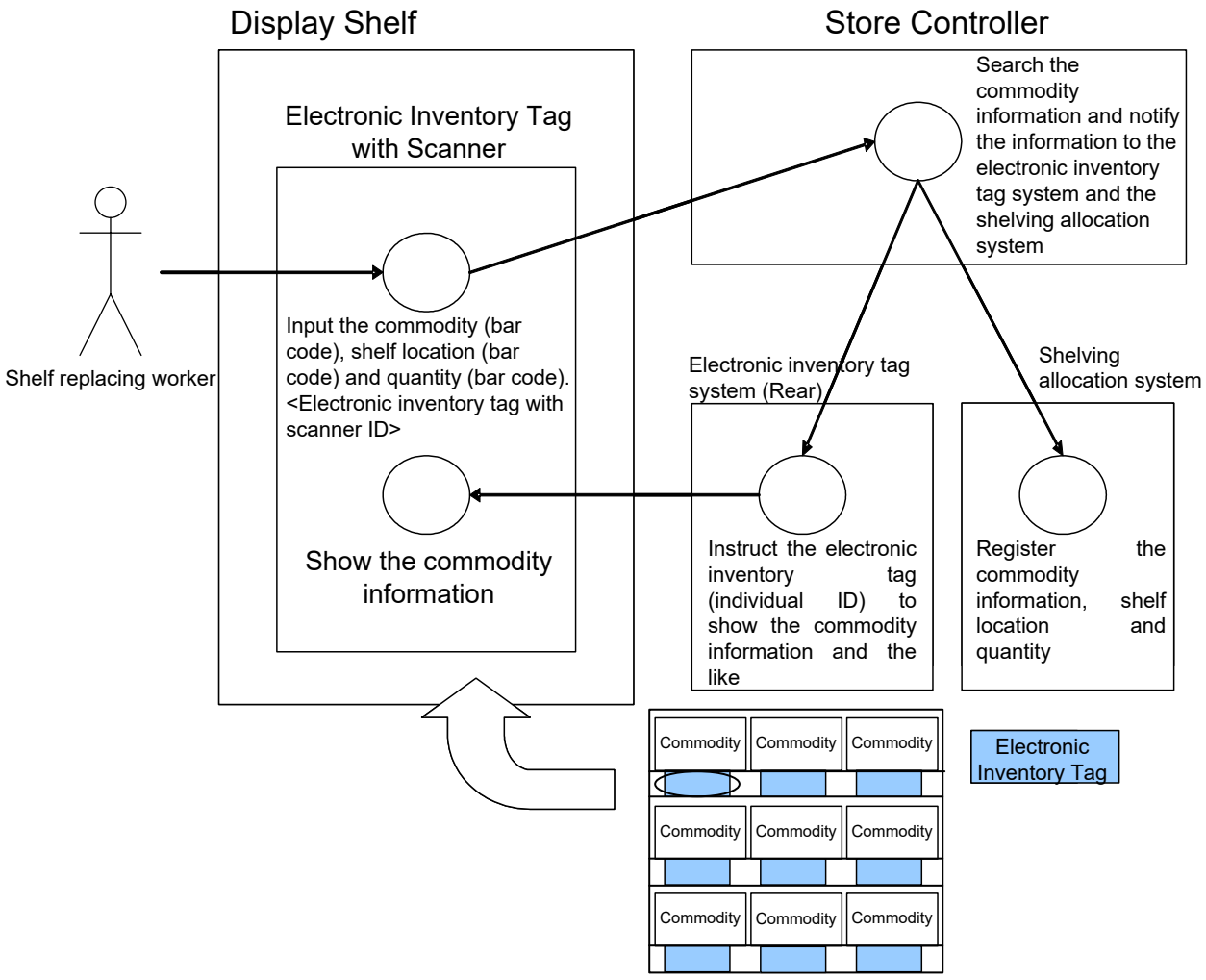
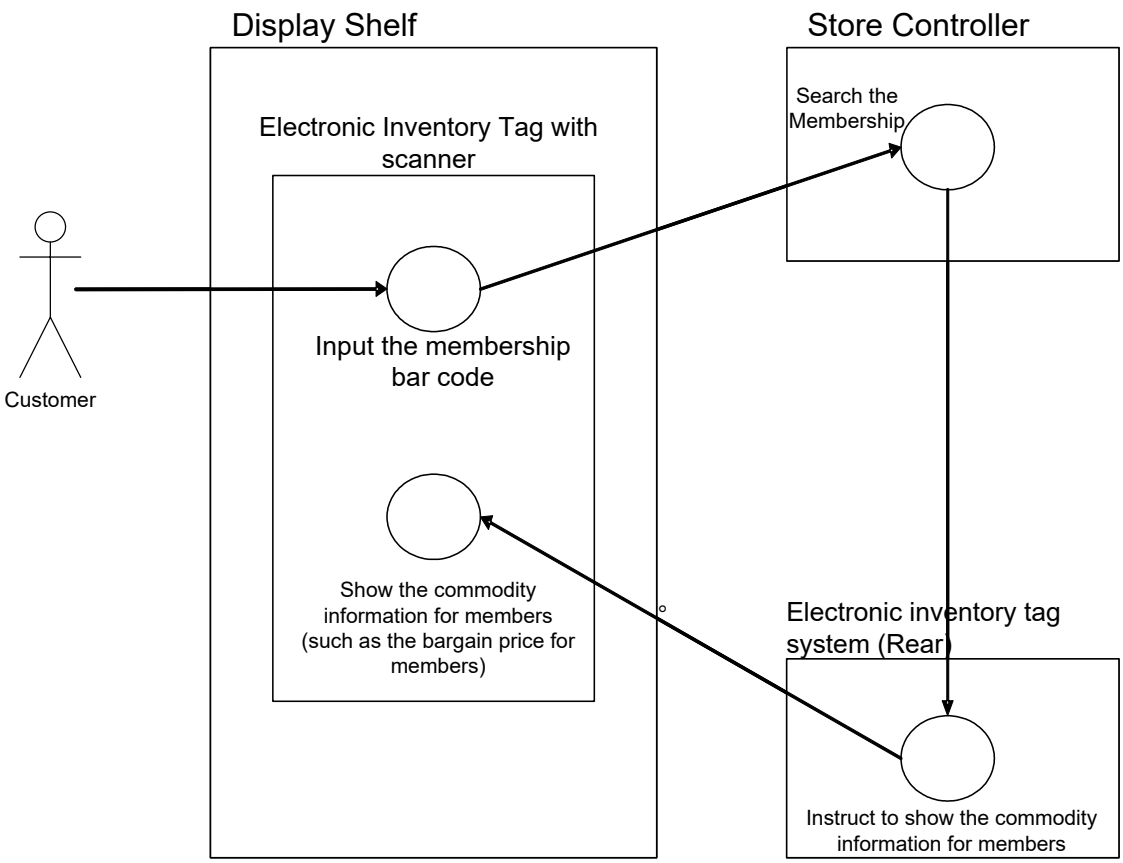


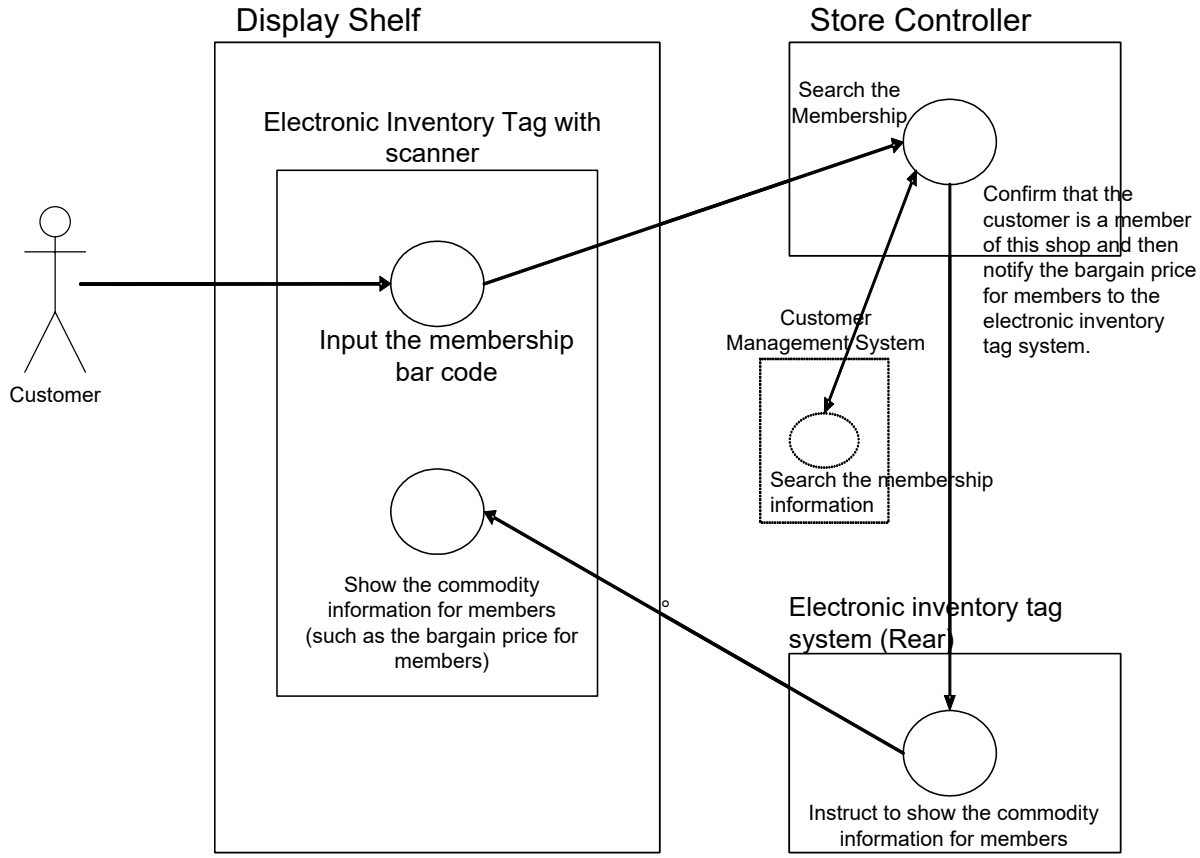
Figure 38: Linkage between Display Shelf and Rear System

Definition of entities (Part 2):

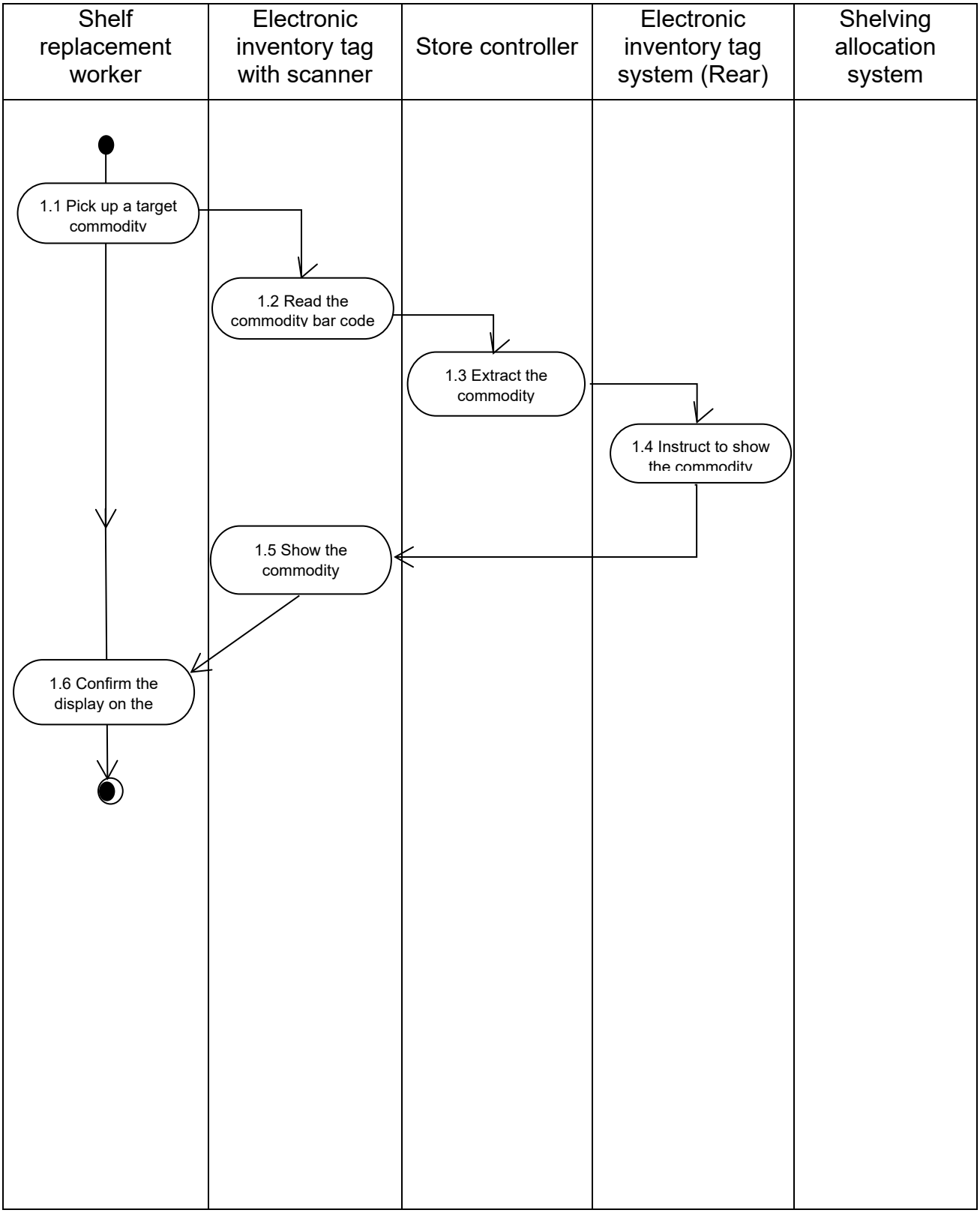
Name	Definition
Customer	A customer is someone who obtains information such as the selling price of the relevant commodity (including the selling price for members) from an electronic inventory tag
Electronic inventory tag with scanner:	An electronic inventory tag with scanner is a device which reads the membership bar code and displays the commodity information such as the commodity price (However, this device does not exist at present).
Store controller	A server which controls commodity information (such as product name, selling price, selling price for members, quantity, and others).
Electronic inventory tag system (Rear):	A server which instructs the electronic inventory tag with scanner to display information and controls the state of display

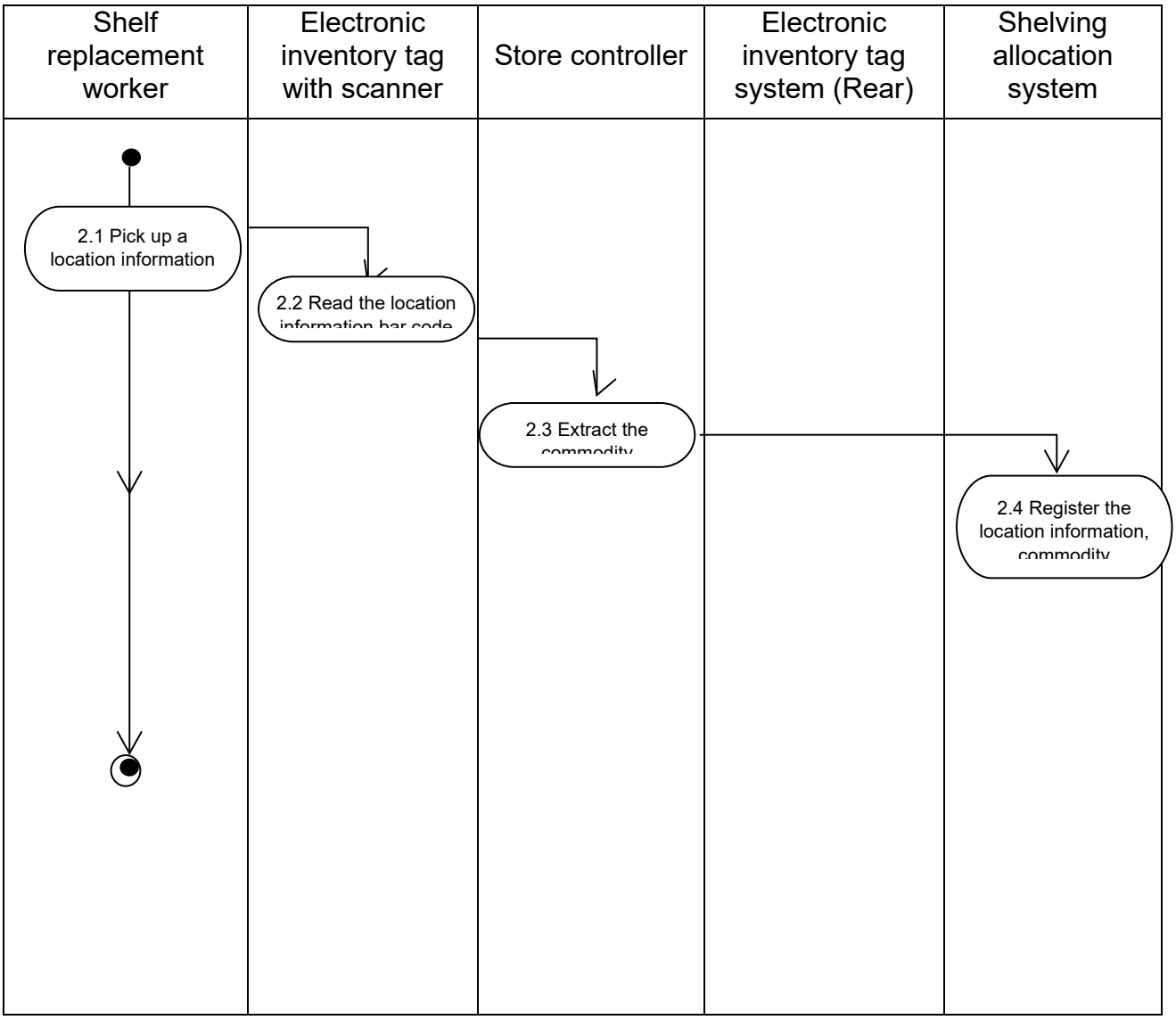

Display Shelf
Store Controller
Search the Membership
Electronic Inventory Tag with scanner
Customer
Input the membership bar code
Show the commodity information for members (such as the bargain price for members)
Electronic inventory tag system (Rear)
Instruct to show the commodity information for members

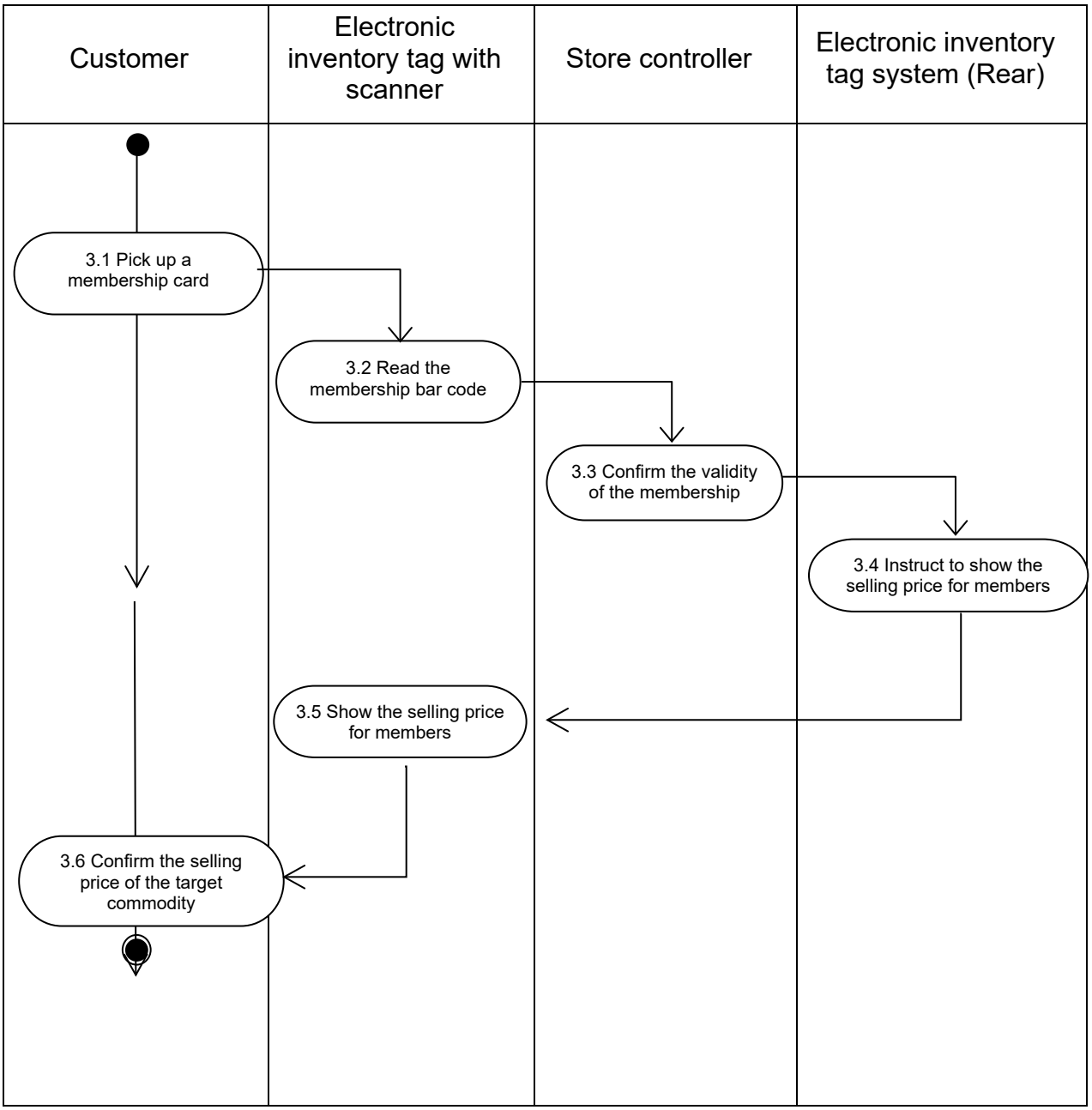
<Alternative method>



Activity Diagram




Shelf replacement worker
Electronic inventory tag with scanner
Store controller
Electronic inventory tag system (Rear)
Shelving allocation system
2.1 Pick up a location information
2.2 Read the location information bar code
2.3 Extract the commodity
2.4 Register the location information, commodity


Customer
Electronic inventory tag with scanner
Store controller
Electronic inventory tag system (Rear)
3.1 Pick up a membership card
3.2 Read the membership bar code
3.3 Confirm the validity of the membership
3.4 Instruct to show the selling price for members
3.5 Show the selling price for members
3.6 Confirm the selling price of the target commodity

3.11 Sub Scope: Cooperation between Electronic Shelf Label and Shelving Allocation Information

Contributor: Retail Science Co., ltd.

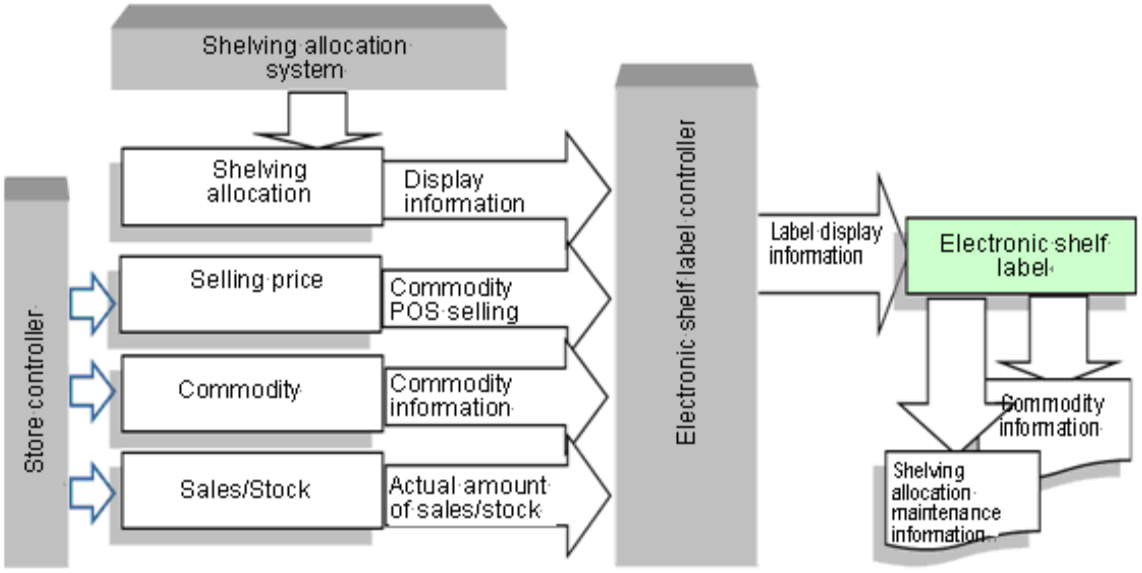


Figure 39: Cooperation between Electronic Shelf Label and Shelving Allocation Information

Target entities

- Shelving allocation (Shelving allocation system)
- Electronic shelf label controller
- Electronic shelf label device
- Shelving allocation maintenance information

Entities not targeted

- Store controller
- Selling price
- Commodity
- Sales/Stock

Stakeholders	Concerns
Shelving Allocation Manager	A shelving allocation manager is in charge of creating and managing display data for each shop. A shelving allocation manager is interested in improving the productivity of a selling space by letting the shop carry out improvement or elimination of commodity correctly according to the plan.

Shop Operator	A shop operator is under an obligation to reflect the shelving allocation planned by the shelving allocation manager on actual shelves of the shop. He/she wishes to do this by carrying out the task efficiently and correctly to the extent possible.

Stakeholders and their concerns

Stakeholders

Definition of Sub Scope

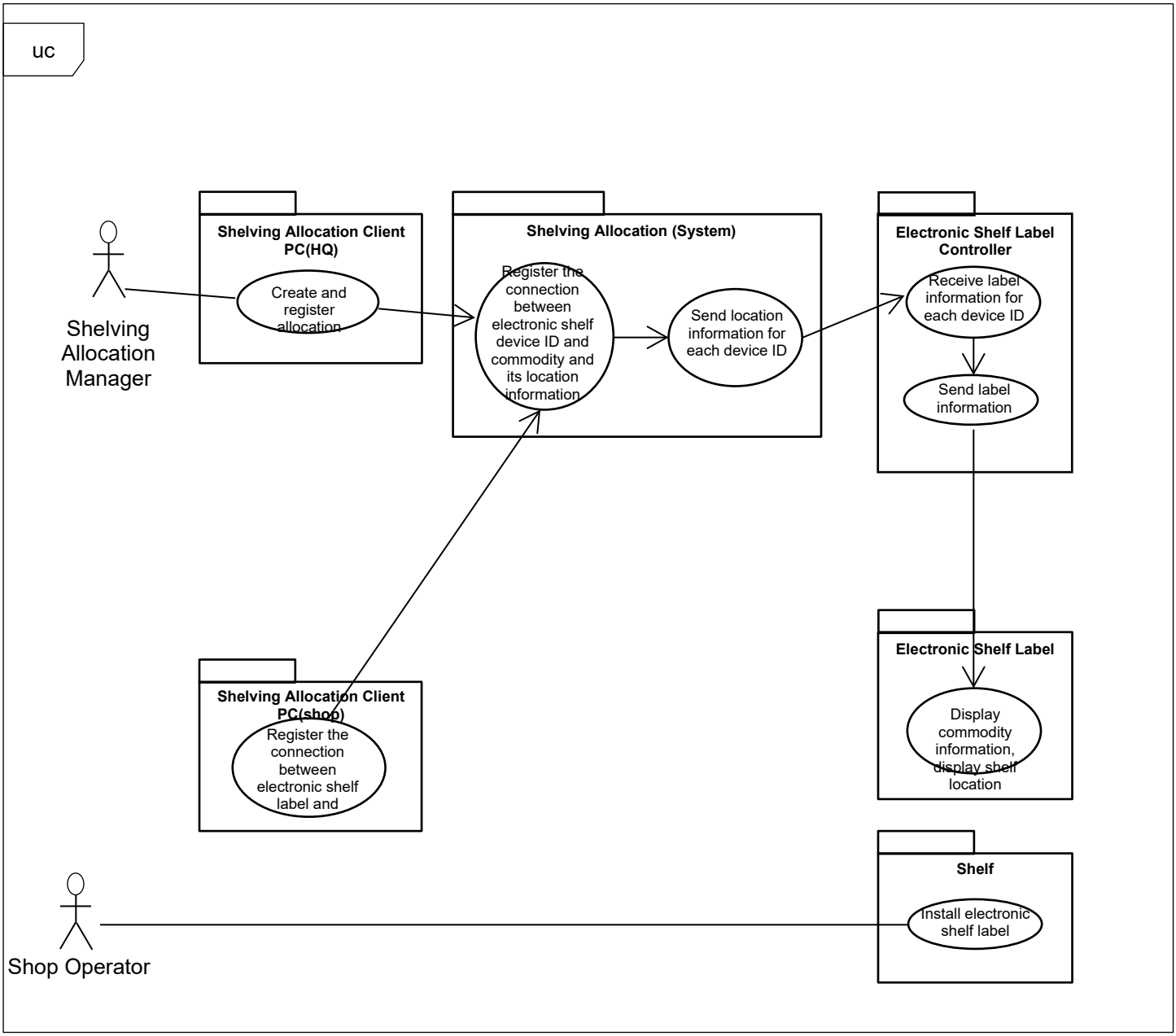
The group of entities related to the electronic shelf label shall be the overall scope. The management of the connection between the location information of an electronic shelf label and the commodity information, an interface between applications or between an application and a device used to display information on electronic shelf labels, and shelving allocation maintenance information displayed on electronic shelf labels shall be sub scope.

Definition of target entities

Name	Definition
Shelving allocation (Shelving allocation system)	An application which imports and maintains shelving allocation information created by an external shelving allocation system or the like for each selling space in each shop of a retailing company and provides a connection between an electronic shelf label device controlled by an electronic shelf label controller and location of a commodity displayed on a shelf of a shop. In some cases, the shelving allocation system is utilized by extending itself. Conventionally, a main requirement for "shelving allocation" is to control commodity codes and location information in shelving allocation, but in this case, the shelving allocation supports electronic shelf label devices installed at a certain location on a shelf in providing commodity information by further controlling serial numbers of the electronic shelf label device, commodity codes, and location information of the commodity. The "shelving allocation" has a trigger to reflect display information to the controller which controls the electronic shelf label device, or, the "shelving allocation" has a controller function in itself to provide information to the electronic shelf label device.
Electronic shelf label controller	This controller controls a serial number of an electronic shelf label device and information it displays. Electronic shelf label controller cooperates with "shelving allocation," and sends information to be displayed on a specified electronic shelf label device to an electronic shelf label. Representative items as information to be maintained are the basic information such as commodity name, price, and ordering unit, and the

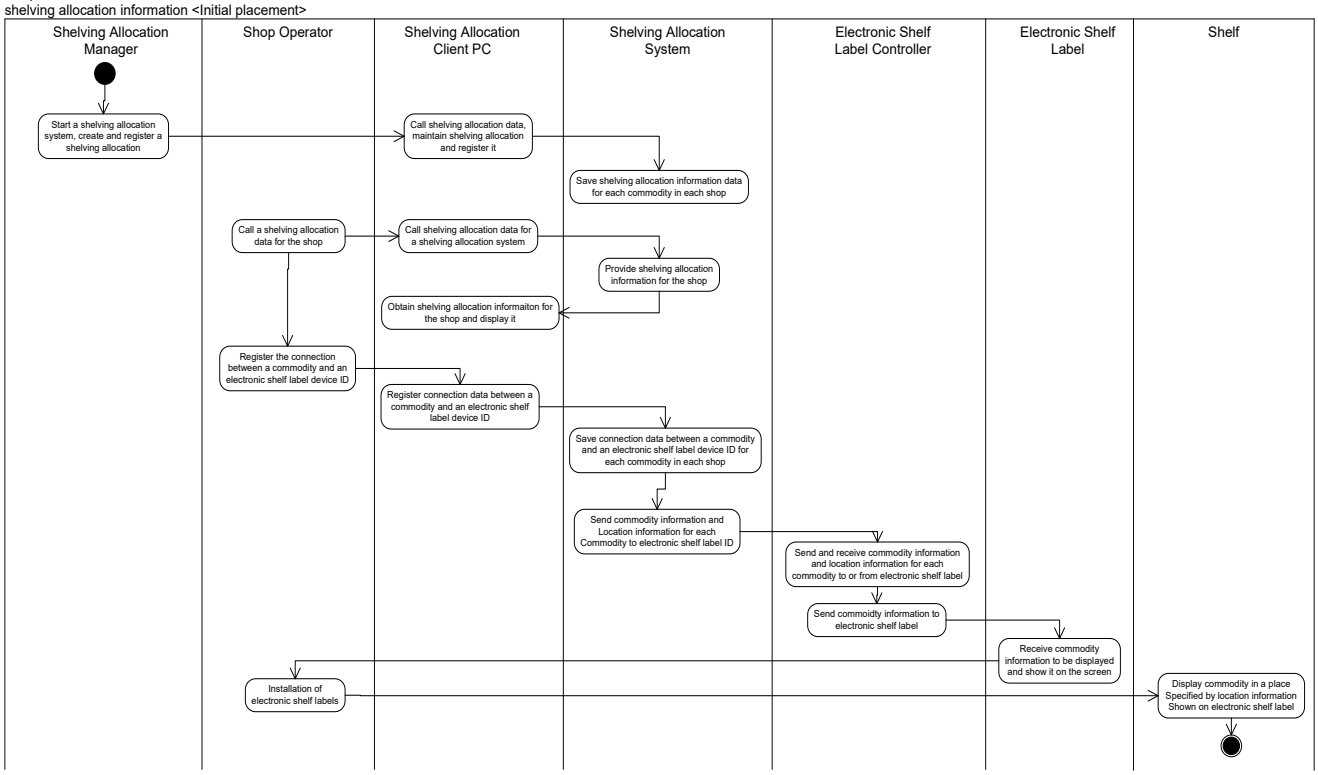
	"shelving allocation maintenance information" used to support a shelving allocation maintenance work, which is a target in this case. According to the performance of electronic shelf label device, in some cases, the electronic shelf label controller may have a control function which is in conformity with a means of communication dedicated to the device. And information such as commodity price is synchronized with master information or the current selling price by a POS system's store controller via LAN.
Electronic shelf label device	This is a device used to display information related to the commodity for customers or employees, and contents to be displayed are controlled by the electronic shelf label controller which manages an electronic shelf label's serial number. It is desirable to use a device which can display the commodity name and bar code on its LCD display or the like in consideration of an objective of this case. And it might be necessary to switch between information for customers and information for employees. As for the means of communication with an electronic shelf label device, it may be advantageous to be able to communicate directly with the device by using wireless LAN connection or Bluetooth connection in the future in consideration of its use for many purposes, but now, it is likely that the use of wired communication or wireless infrared communication via an electronic shelf label controller dedicated for the device will spread first in consideration of aspects such as the cost and popularity.
Shelving allocation maintenance information	Shelving allocation maintenance information is information related to shelving allocation maintenance displayed on the electronic shelf label device. The commodity name, price, and commodity code (bar code) must be displayed on the electronic shelf label as an explanation of the commodity, but in addition to them, in this scope, information related to the shelving allocation maintenance, which is necessary for employees, (display location information (gondola number, shelf number, display order), number of faces, applicable period for the commodity) are also required.

Activity Diagram

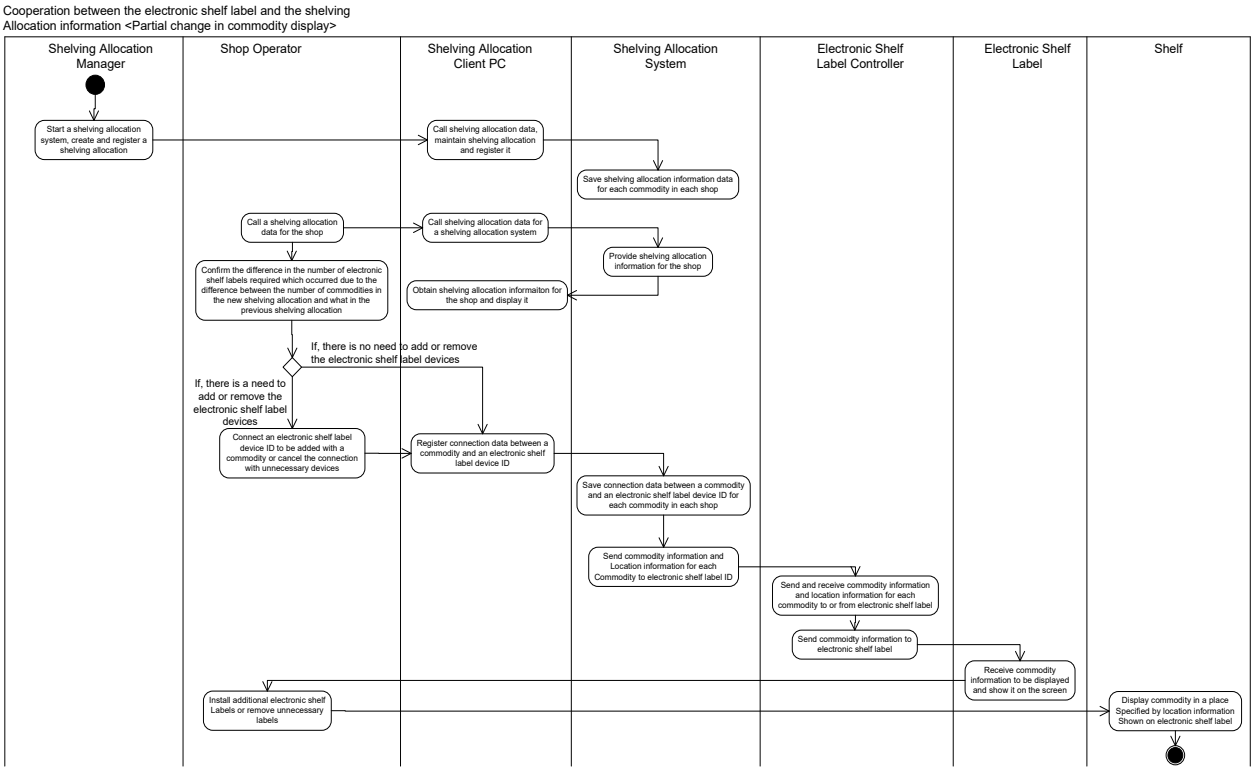


Activity Diagram (High level use case)

Cooperation between the electronic shelf label and the shelving allocation information <Initial placement>



Activity Diagram (Extended use case 1)



3.12 Sub Scope: Self-Service Refueling

Contributor: NEC Infrontia Corporation

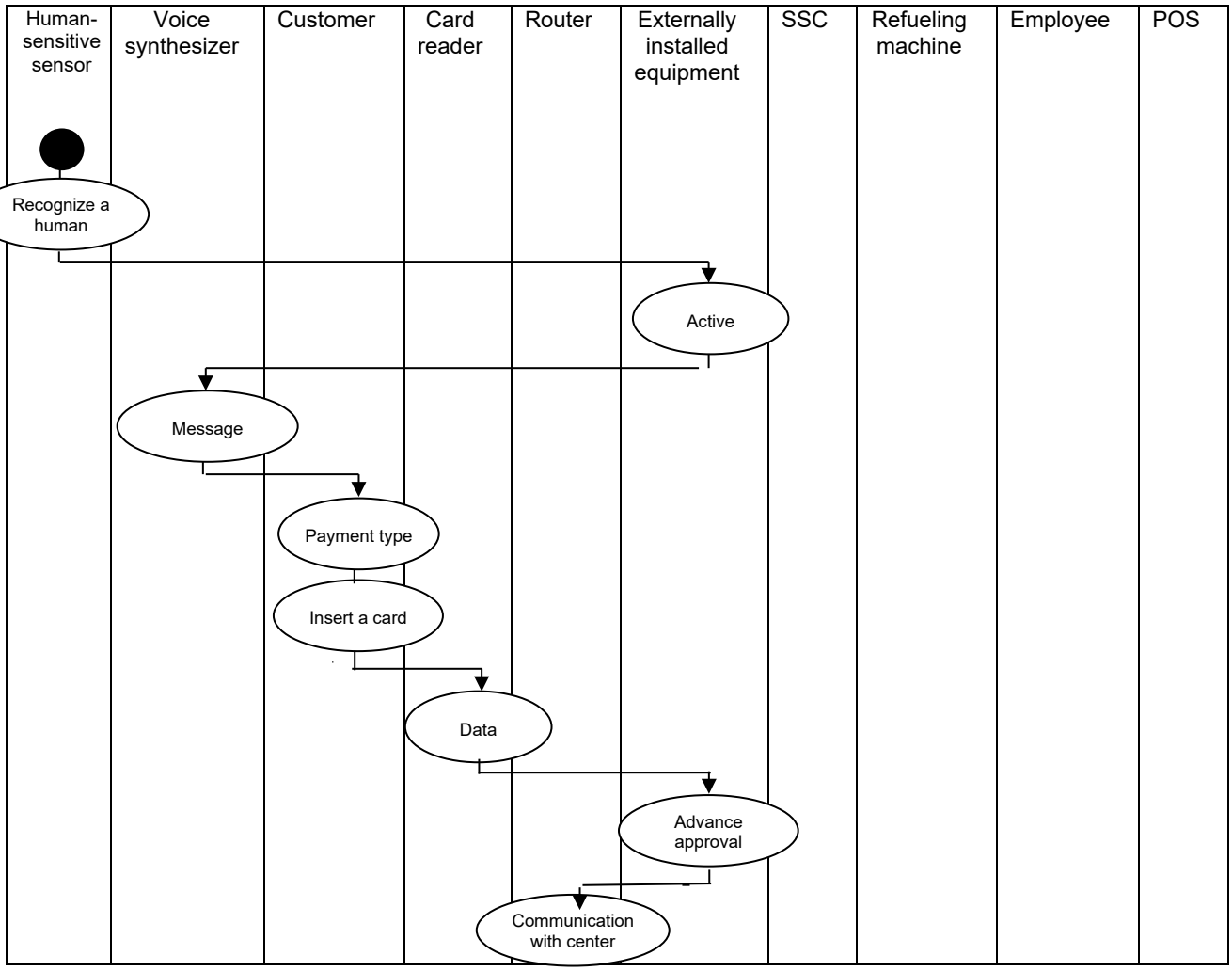
Perform a self-service refueling transaction at a SS (Service Station).

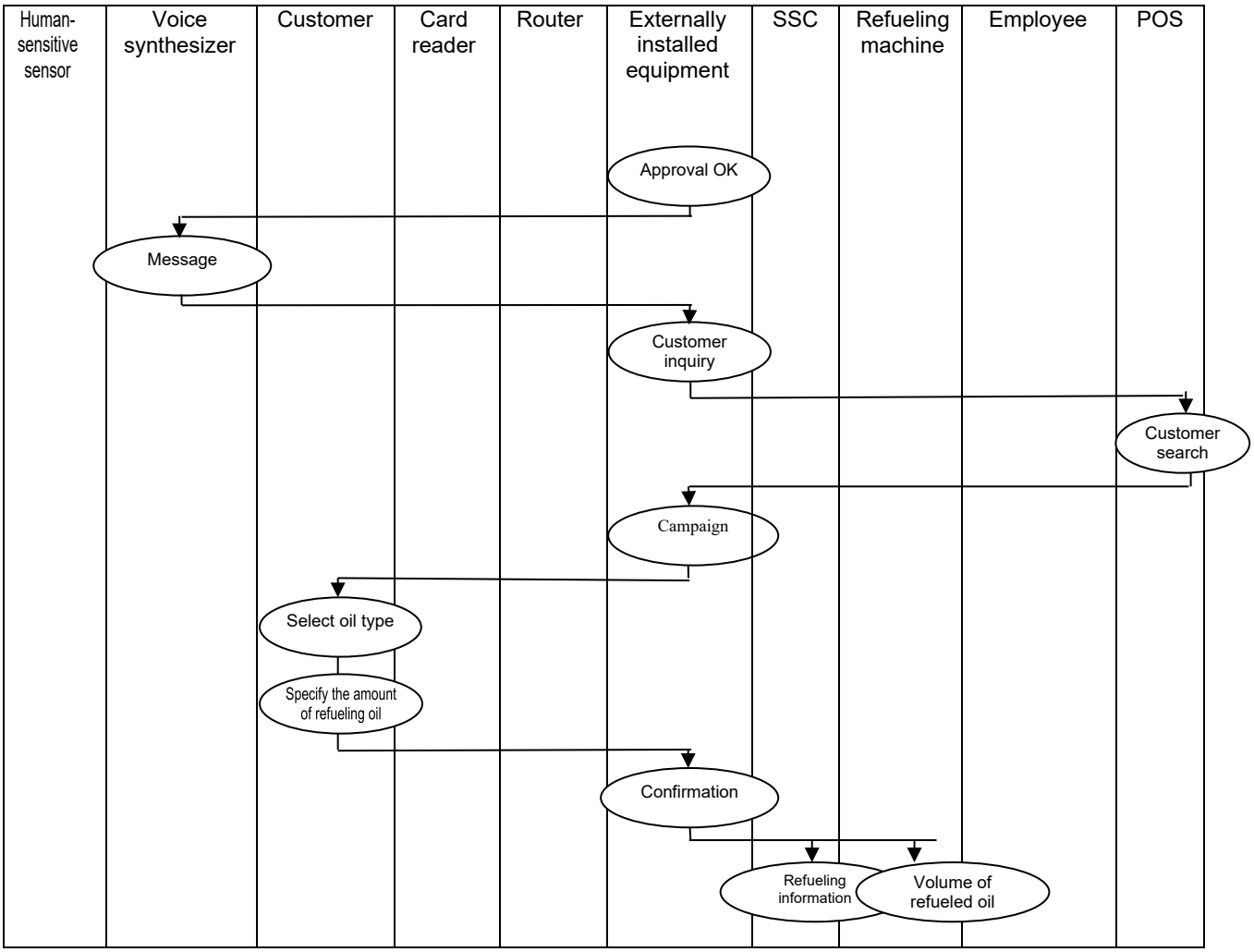
Definition of entities:

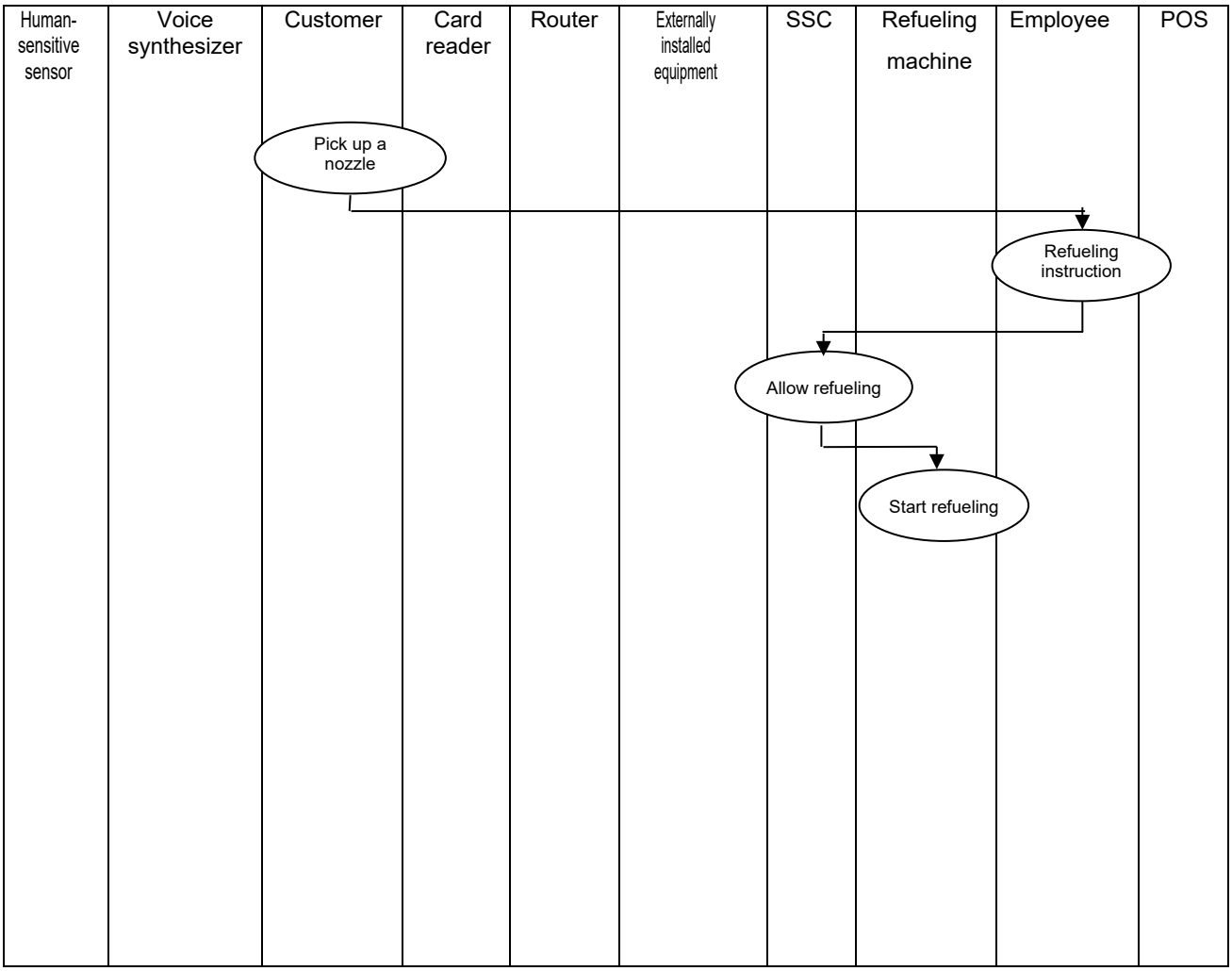
Name	Definition
Customer	Performs a self-service refueling at a SS
Car	Refueled
Employee	In response to the refueling transaction by the customer, he/she allows refueling and stops refueling in a case of emergency through SSC
Credit card	Performs a refueling transaction using a credit card
Electronic money	Performs a refueling transaction using electronic money
Cash transaction	Performs a refueling transaction using cash
SSC (Self Service Console)	Controls refueling equipment by allowing or prohibiting a refueling
POS	Performs a refueling transaction, ordering, stocking, single item sales, card issuance, customer management, and commodity management

Card reader	Reads and issues magnetic cards
Contactless reader	Handles electronic money
Human-sensitive sensor	Activates externally installed equipment when someone approaches it
Voice synthesizer	Lets the customer know what to do next with voice
Printer	Prints a receipt when a refueling is completed
Payment machine	Puts money in it for settlement in the case of cash transaction
Change machine	Dispenses the change when a refueling transaction is completed
Externally installed equipment	Specifies with it the type of settlement, type of oil, and amount of oil outdoors. Performs an advance approval process in the case of credit card settlement. Performs a credit confirmation in the case of electronic settlement. Performs a money-receiving process and instructs the change machine to dispense the change in the case of cash settlement. Outputs a receipt through the printer when a refueling is completed
Measuring machine	Measures the amount of oil refueled
Refueling machine	Refuels cars
Oil level indicator	Measures the remaining amount of each type of oil in tank
Car-washing machine	Washes cars
Router	Performs communication with a credit center and a totalization center
Forecourt Device Communication BOX	Acts as a bridge between POS, externally installed equipment, SSC, and Forecourt devices
Shop server	Performs customer management, outputting of sales management report, and exchanging information with the primary distributor
Customer information	Information related to a car (Maintenance, vehicle inspection, oil change) Management of information related to a customer (Hobby, address, family makeup, age, etc.)
Transaction data	Amount of refueled oil, payment type, unit price.

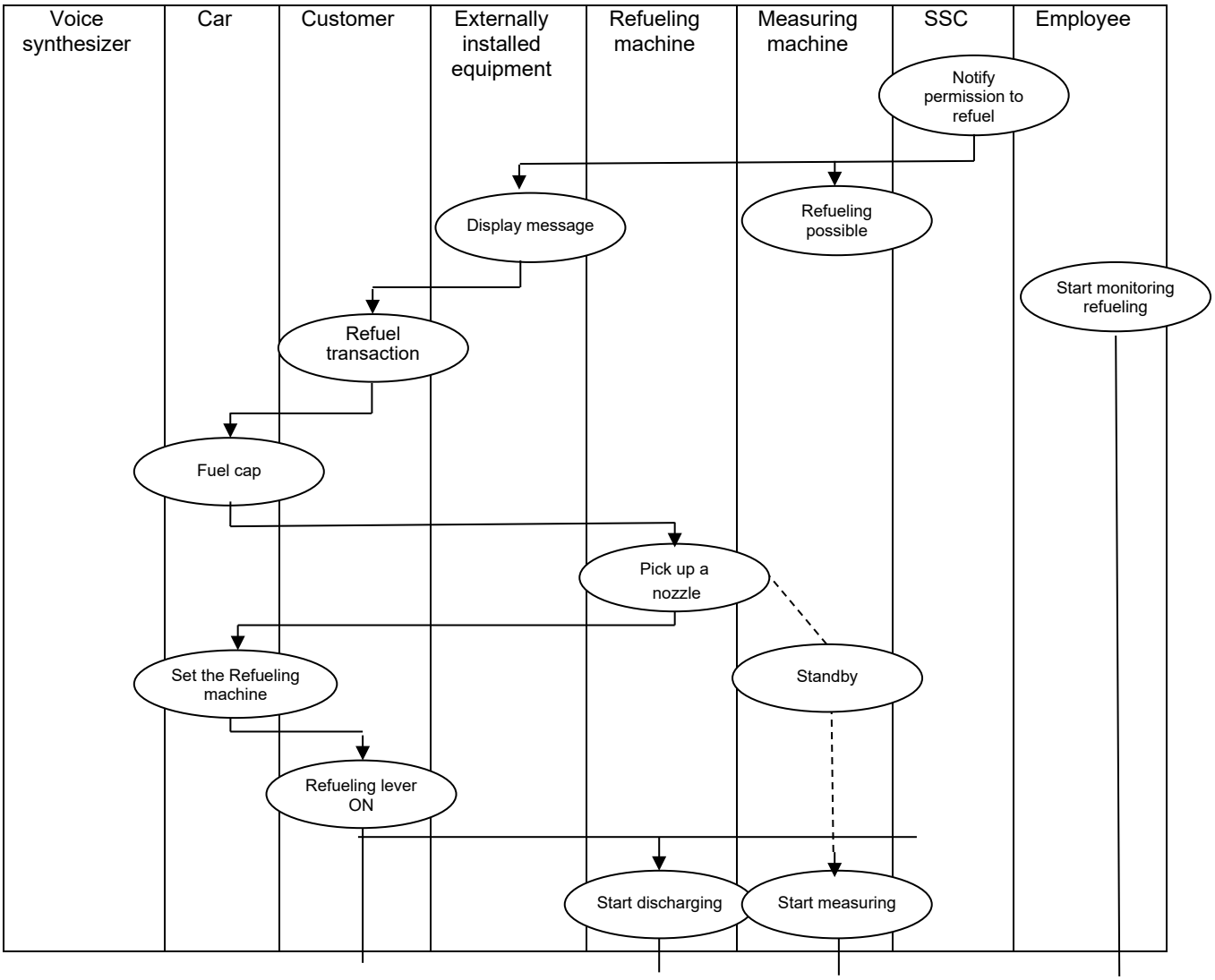
Activity Diagram

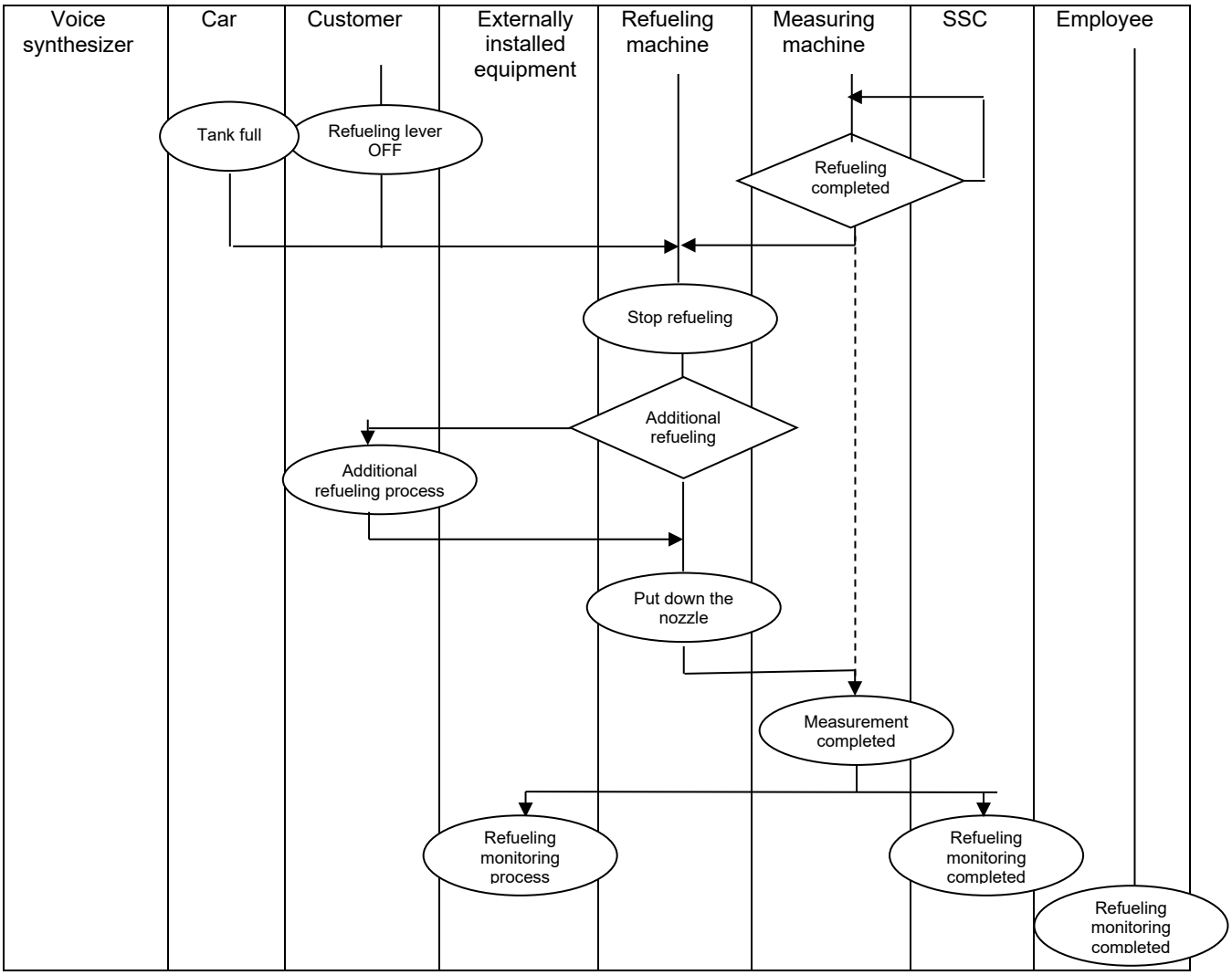



Human-sensitive sensor
Voice synthesizer
Customer
Card reader
Router
Externally installed equipment
SSC
Refueling machine
Employee
POS
Approval OK
Message
Customer inquiry
Customer search
Campaign
Select oil type
Specify the amount of refueling oil
Confirmation
Refueling information
Volume of refueled oil

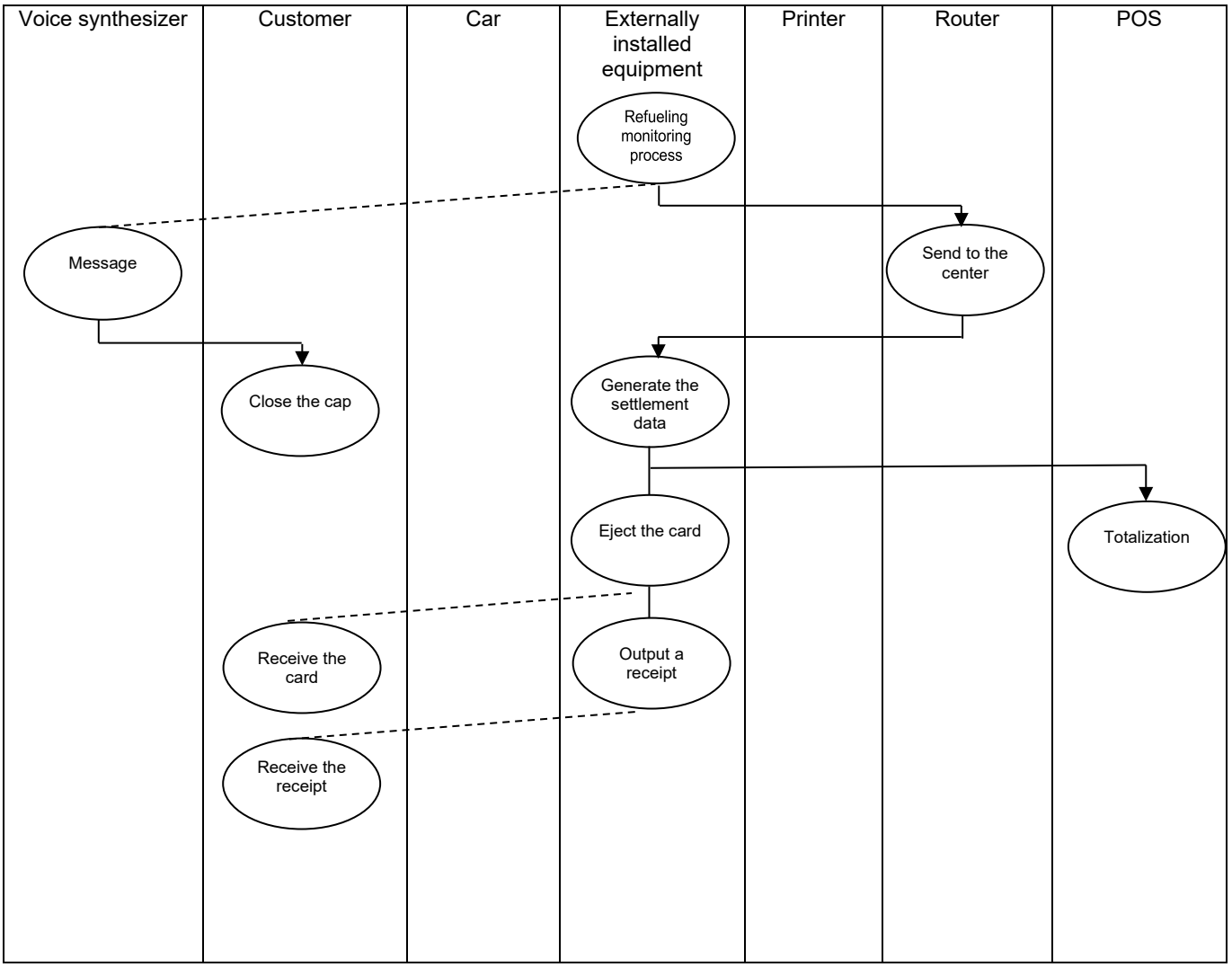

Human-sensitive sensor
Voice synthesizer
Customer
Card reader
Router
Externally installed equipment
SSC
Refueling machine
Employee
POS
Pick up a nozzle
Refueling instruction
Allow refueling
Start refueling

Activity Diagram




Voice synthesizer
Car
Customer
Externally installed equipment
Refueling machine
Measuring machine
SSC
Employee
Tank full
Refueling lever OFF
Refueling completed
Stop refueling
Additional refueling
Additional refueling process
Put down the nozzle
Measurement completed
Refueling monitoring process
Refueling monitoring completed
Refueling monitoring completed

Activity Diagram



4. DOCUMENT HISTORY

Version History

Ver	Date	Sections	Description of Change
0.1	2007-09-02	All	Initial document creation
0.2	2007-09-11	All	Fixed references, reworded, split components between WS and other
0.3	2008.07.31	All	Reorganized doc, added notes about what sections should contain
0.4	2008.08.26	Security, Flow	Added specific security requirements, WS-POS Flow section, Out-of-scope
0.5	2008.09.24	All	Events, Security, Service Description, Discovery
1.1	2011.05.15	All	Updated document to reflect inputs from field beta testing provided by OPOS-J; updated support for WSDL generation using WCF and JAX-WS and revised XMLPOS to ensure interoperability; updated documentation for clarity; added WS-POS class diagram .Net generated examples for all UnifiedPOS 1.14 peripheral devices. Added software support files to aide developers.
1.2	2013.01.22	*As Noted*	Version 1.2 of this specification contains several new chapters and updates to existing chapters that provide clarifications and necessary functional enhancements to Version 1.1. These are detailed below, with links to the corresponding pages and/or chapters as appropriate. • Updated the Version and issue date on the front page. • Updated Web Service Components Page 9 • Updated chapter “Service Description and Discovery” starting on Page 14. • Added a new chapter describing the “WS-POS Behavior Models” starting on Page 18. • Added a new chapter describing the “Introduction to Properties, Methods and Events” starting on Page 18. • Added a new chapter describing the “WS-POS Communication Model” on Page 19. • Added a new chapter describing the “Session Management and Device Control” on Page 20. • Added a new chapter describing the “Introduction of WS-POS Session Manager concept” on Page 20. • Added a new chapter describing the “Identifying WS-POS Session” on Page 21. • Added a new chapter describing the “Typical sequence to establish WS-POS session” on Page 22. • Added a new chapter describing the “Calling WS-POS Service Methods and Using Properties” on Page 23.
1.2		*As Noted*	

	2013. 01.22

			associated chapter.Update version and release date in the front page • Updated “Overview” chapter on Page 8 • Added a new chapter describing “Security Consideration” on Page 21 • Updated “WS-POS Methods and Device Methods” chapter on Page 29 • Added a new chapter describing “Resolution of frequent communication events in the Event notification” on Page 38 • Updated “WS-POS Method References (UPOS UML Style)” chapter on Page 58 • Updated “WS-POS Event Reference in Bi-Directional Communication” chapter on Page 76 • Updated “Modifications to XMLPOS” chapter on Page 82 • Added a new chapter describing “Retrieving consumerID that has been successfully claimed” on Page 88 • Updated “Security” chapter on Page 108 • Added a new chapter describing “Software Support Files” on Page 168 • Replaced “UPOS” to “UnifiedPOS” in this specification • Added getBinaryConversion and setBinaryConversion method on Page 30, 68, 69, 82, 82
1.3.1	2018. 02.06	*all*	Version 1.3.1, released in 2018, represents only a migration of this specification from the National Retail Federation (NRF) to the Object Management Group (OMG) through an extensive agreement. All Copyright ownership is transferred to OMG under this agreement. This version includes the replacement of the copyright statements and minor text edits to accommodate this transition. The changes are detailed below. No other changes to other sections of the standard were made and remain the same as in Version 1.3. • Updated the Version and issue date on the front. • Updated the Copyright and Disclaimer notices. • Updated the Version and member in “OPOS-J Work Team For WS-POS Version 1.3.1”. • Updated the Table of Contents to reflect additional edits. • Updated the version in “Software Support Files from 1.3 to 1.3.1

5. REFERENCED DOCUMENTS AND SOFTWARE SUPPORT FILES

5.1 Referenced Documents

Term	Definition
WS-I Basic Profile V1.2	http://www.ws-i.org/Profiles/BasicProfile-1.2.html
SOAP 1.1	http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
SOAP HTTP Binding	http://www.w3.org/TR/2007/REC-soap12-part2-20070427/#soapinhttp
SOAP HTTP Status Codes, Error Conditions	http://www.w3.org/TR/2007/REC-soap12-part2-20070427/#http-reqbindwaitstate
WS-Addressing 1.0	http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/
WSDL 1.1	http://www.w3.org/TR/wsdl
WS-MetadataExchange Publication	http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf
UDDI V3	http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm
TLS V1.2	https://www.ietf.org/rfc/rfc5246.txt
WS-Security V1.1	http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
Using WSDL in a UDDI Registry, Version 2.0.2	http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm
UnifiedPOS 1.14 .1	www.omg.org
ARTS XML Best Practices	www.omg.org

5.2 Software Support Files

The following software SDK files are available (See: http://www.omg.org/retail/unified-pos.htm) to aid Developers in the creation of WS-POS compliant applications.

WS-POS Ver. 1.1 File Lists

1) WS-POS 1.1 equivalent XML Schema Files (XMLPOS 1.13 is used)

   File Name: WS-POS1.1XSD.zip

2) WS-POS 1.1 equivalent WSDL Files (Independent files and implementation dependent files)

   File Name: WS-POS1.1WSDL.zip

3) WS-POS 1.1 WCF Contract Files

   File Name: WS-POS1.1WCFContract.zip

4) WS-POS 1.1 JAX-WS Contract Files

File Name: WS-POS1.1JAX-WSContract.zip

5) WS-POS 1.1 WCF Sample Program

File Name: WS-POS1.1WCFSample.zip

6) WS-POS 1.1 JAX-WS Sample Program

File Name: WS-POS1.1JAX-WSSample.zip

***Important Notice For Condition of Use**:*

***OMG** and **User** agree that the Software is provided "**AS IS**" and that **OMG** makes no warranty as to the Software. **OMG** DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, RELATED TO THE SOFTWARE, ITS USE OR ANY INABILITY TO USE IT, THE RESULTS OF ITS USE AND THIS AGREEMENT.*

5.3 Software Support Files *Added in Version 1.2*

The following software SDK files are available (See: http://www.omg.org/retail/unified-pos.htm) to aid Developers in the creation of WS-POS 1.2 compliant applications.

WS-POS Ver. 1.2 File Lists

1) WS-POS 1.2 equivalent XML Schema Files (XMLPOS 1.14*** is used)

File Name: WS-POS1.2XSD.zip

2) WS-POS 1.2 equivalent WSDL Files (Independent files and implementation dependent files)

File Name: WS-POS1.2WSDL.zip

3) WS-POS 1.2 WCF Contract Files

File Name: WS-POS1.2WCFContract.zip

4) WS-POS 1.2 JAX-WS Contract Files

File Name: WS-POS1.2JAX-WSContract.zip

5) WS-POS 1.2 Class Diagrams

****Note: WS-POS 1.2 is designed to support UnifiedPOS 1.14 which incorporates XMLPOS 1.14. However, WS-POS 1.2 will also support previous versions of UnifiedPOS 1.13 and 1.12 where the extended functionality in UnifiedPOS 1.14 may not be needed or support for XMLPOS 1.14, POS For .Net 1.14, Common Control Objects 1.14, or JavaPOS Services 1.14 is not available. It is strongly suggested when given a choice, UnifiedPOS 1.14 support be implemented.*

***Important Notice For Condition of Use**:*

***OMG** and **User** agree that the Software is provided "**AS IS**" and that **OMG** makes no warranty as to the Software. **OMG** DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, RELATED TO THE SOFTWARE, ITS USE OR ANY INABILITY TO USE IT, THE RESULTS OF ITS USE AND THIS AGREEMENT.*

5.4 Software Support Files *Added in Version 1.3*

The following software SDK files are available (See: http://www.omg.org/retail/unified-pos.htm) to aid Developers in the creation of WS-POS 1.3 compliant applications.

WS-POS Ver. 1.3 File Lists

1) WS-POS 1.3 equivalent XML Schema Files (XMLPOS for WS-POS 1.3***)

File Name: WS-POS1.3XSD.zip

2) WS-POS 1.3 equivalent WSDL Files (Independent files and implementation dependent files)

File Name: WS-POS1.3WSDL.zip

3) WS-POS 1.3 WCF Contract Files

File Name: WS-POS1.3WCFContract.zip

4) WS-POS 1.3 JAX-WS Contract Files

File Name: WS-POS1.3JAX-WSContract.zip

5) WS-POS 1.3 Class Diagrams

****Note: XMLPOS for WS-POS 1.3 is XML schema file created with consideration of compatibility to XMLPOS1.14.1 and previous version of WS-POS.*

5.5 Software Support Files *Updated in Version 1.3.1*

The following software SDK files are available (See:**Error! Hyperlink reference not valid.** https://www.omg.org/spec/WS-POS/) to aid Developers in the creation of WS-POS 1.3.1 compliant applications.

WS-POS Ver. 1.3.1 File Lists

1) WS-POS 1.3.1 equivalent XML Schema Files (XMLPOS for WS-POS 1.3.1***)

File Name: WS-POS1.3.1XSD.zip

2) WS-POS 1.3.1 equivalent WSDL Files (Independent files and implementation dependent files)

File Name: WS-POS1.3.1WSDL.zip

3) WS-POS 1.3.1 WCF Contract Files

File Name: WS-POS1.3.1WCFContract.zip

4) WS-POS 1.3.1 JAX-WS Contract Files

File Name: WS-POS1.3.1JAX-WSContract.zip

5) WS-POS 1.3.1 Class Diagrams

File Name: WS-POSVer.1.3.1_Technical_Specification_ClassDiagram.docx

****Note: XMLPOS for WS-POS 1.3.1 is XML schema file created with consideration of compatibility to XMLPOS1.14.1 and previous version of WS-POS.*

***Important Notice For Condition of Use**:*

***OMG** and **User** agree that the Software is provided "**AS IS**" and that **OMG** makes no warranty as to the Software. **OMG** DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, RELATED TO THE SOFTWARE, ITS USE OR ANY INABILITY TO USE IT, THE RESULTS OF ITS USE AND THIS AGREEMENT.*

6. WS-POS CLASS DIAGRAMS *Updated in Version 1.2*

Beginning in version 1.2, the examples of the Class Diagrams that describe the WS-POS services that support the thirty-six POS peripheral devices that are described in the UnifiedPOS specification have been moved to the software support files noted in section 6.3. They are included to help in the understanding of the interrelationship of the processes and data that is necessary for a web services implementation.

Important Note Concerning the UML Examples

The included class diagram examples were distilled from C# code using the Microsoft Visual Studio 2010 development tools. As a result, the programmatic dependent language features are valid for C# only. A similar set of Class Diagrams for a Java implementation could also be generated using an equivalent Java development tool but the programmatic dependent language features would be different from what is shown.

The intent is to show the relative relationship of the UnifiedPOS defined Properties, Methods, Events, and Data Parameters using standard UML imagery. The UnifiedPOS documentation should be consulted for detailed Peripheral Device operational information.

The peripheral devices are shown in alphabetical order and equate to the respective peripheral device chapters in the UnifiedPOS standard.

The following provides an example of how a WS-POS mapping schema would work to resolve the UnifiedPOS Properties, Methods, and Events to WSDLs, C# web service contracts, and Java web service contracts.

First the example shows how the read-write property "**DeviceEnable**" and the method "**checkHealth**" for the UnifiedPOS device category "Belt" would be represented as WSDL document.

Second the corresponding Java and C# classes will be generated by using JAX-WS and WCF tools. All other properties, methods for the Belt are omitted (by '**…**') for better readability.

Note that the Events are specified in separate WSDL files and are not included here in this example.

/*---------- BeltV1.2.0.wsdl start ----------*/

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
                  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                  xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
                  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:tns="http://www.nrf-arts.org/UnifiedPOS/Belt/"
                  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
                  xmlns:wsa10="http://www.w3.org/2005/08/addressing"
                  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
                             name="BeltService"
                             targetNamespace="http://www.nrf-arts.org/UnifiedPOS/Belt/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
          <xsd:schema targetNamespace="http://www.nrf-arts.org/UnifiedPOS/Belt/Imports">
          <xsd:import namespace="http://www.nrf-arts.org/UnifiedPOS/Belt/"
schemaLocation="BeltV1.14.xsd" />
     </xsd:schema>
  </wsdl:types>

…
<wsdl:message name="Belt_GetDeviceEnabled_InputMessage">
    <wsdl:part name="parameters" element="tns:GetDeviceEnabled" />
</wsdl:message>

  <wsdl:message name="Belt_GetDeviceEnabled_OutputMessage">
    <wsdl:part name="parameters" element="tns:GetDeviceEnabledResponse" />
  </wsdl:message>

  <wsdl:message name="Belt_GetDeviceEnabled_UposException_FaultMessage">
    <wsdl:part name="detail" element="tns:UposException" />
  </wsdl:message>

  <wsdl:message name="Belt_SetDeviceEnabled_InputMessage">
    <wsdl:part name="parameters" element="tns:SetDeviceEnabled" />
  </wsdl:message>

  <wsdl:message name="Belt_SetDeviceEnabled_OutputMessage">
    <wsdl:part name="parameters" element="tns:SetDeviceEnabledResponse" />
  </wsdl:message>
```

```
  <wsdl:message name="Belt_SetDeviceEnabled_UposException_FaultMessage">
    <wsdl:part name="detail" element="tns:UposException" />
  </wsdl:message>

...
  <wsdl:message name="Belt_CheckHealth_InputMessage">
    <wsdl:part name="parameters" element="tns:CheckHealth" />
  </wsdl:message>

  <wsdl:message name="Belt_CheckHealth_OutputMessage">
    <wsdl:part name="parameters" element="tns:CheckHealthResponse" />
  </wsdl:message>

  <wsdl:message name="Belt_CheckHealth_UposException_FaultMessage">
    <wsdl:part name="detail" element="tns:UposException" />
  </wsdl:message>

 ...
<wsdl:portType name="Belt">

...
  <wsdl:operation name="GetDeviceEnabled">

       <wsdl:input wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabled"
          message="tns:Belt_GetDeviceEnabled_InputMessage" />

      <wsdl:output wsaw:Action="http://www.nrf-
arts.org/UnifiedPOS/Belt/GetDeviceEnabledResponse"
          message="tns:Belt_GetDeviceEnabled_OutputMessage" />

      <wsdl:fault wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/UposException"
name="UposException"
          message="tns:Belt_GetDeviceEnabled_UposException_FaultMessage" />

    </wsdl:operation>

    <wsdl:operation name="SetDeviceEnabled">

      <wsdl:input wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabled"
          message="tns:Belt_SetDeviceEnabled_InputMessage" />

      <wsdl:output wsaw:Action="http://www.nrf-
arts.org/UnifiedPOS/Belt/SetDeviceEnabledResponse"
          message="tns:Belt_SetDeviceEnabled_OutputMessage" />

      <wsdl:fault wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/UposException"
name="UposException"
          message="tns:Belt_SetDeviceEnabled_UposException_FaultMessage" />

    </wsdl:operation>

...
    <wsdl:operation name="CheckHealth">

      <wsdl:input wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealth"
           message="tns:Belt_CheckHealth_InputMessage" />

      <wsdl:output wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealthResponse"
           message="tns:Belt_CheckHealth_OutputMessage" />

      <wsdl:fault wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/UposException"
name="UposException"
           message="tns:Belt_CheckHealth_UposException_FaultMessage" />
    </wsdl:operation>

...
  </wsdl:portType>

</wsdl:definitions>
```

/*---------- BeltV1.1.0.wsdl end ----------*/

Using the WCF tool “svcutil.exe” the following WCF contract in C# would be generated from:

/*--------- Belt.cs start ----------*/

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace UnifiedPOS.Belt
{
    [ServiceContract(Namespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")]
    public interface Belt
    {
                    …

        [OperationContract(Action = "http://www.nrf-
arts.org/UnifiedPOS/Belt/GetDeviceEnabled",
            ReplyAction = "http://www.nrf-
arts.org/UnifiedPOS/Belt/GetDeviceEnabledResponse")]

        [FaultContract(typeof(UposException), Action = "http://www.nrf-
arts.org/UnifiedPOS/Belt/UposException", Name = "UposException")]
            bool GetDeviceEnabled();

        [OperationContract(Action = "http://www.nrf-
arts.org/UnifiedPOS/Belt/SetDeviceEnabled",
            ReplyAction = "http://www.nrf-
arts.org/UnifiedPOS/Belt/SetDeviceEnabledResponse")]

        [FaultContract(typeof(UposException), Action = "http://www.nrf-
arts.org/UnifiedPOS/Belt/UposException", Name = "UposException")]
            void SetDeviceEnabled(bool DeviceEnabled);


                    …

        [OperationContract(Action = "http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealth",
            ReplyAction = "http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealthResponse")]

        [FaultContract(typeof(UposException), Action = "http://www.nrf-
arts.org/UnifiedPOS/Belt/UposException", Name = "UposException")]
            void CheckHealth(HealthCheckLevel Level);


                    …
    }


                    …
    [DataContract(Namespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")]


  public enum HealthCheckLevel
    {
        [EnumMember]
           External,
        [EnumMember]
           Interactive,
        [EnumMember]
           Internal,
    }


            …
}
```

/*--------- Belt.cs end ----------*/

For Java the JAX-WS contract generated with the JAX-WS tool “wsimport” would be look like the following:

/*--------- Belt.java start ----------*/

```
package org.arts.unifiedpos.belt;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.xml.ws.Action;
import javax.xml.ws.FaultAction;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;


@WebService(name = "Belt", targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")
public interface Belt
    {


     ...
        @Action(input = "http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabled",
              output = "http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabledResponse",
               fault =
                 {
                   @FaultAction(className = org.arts.unifiedpos.belt.FaultException.class,
                                       value = "http://www.nrf-
arts.org/UnifiedPOS/Belt/UposException")
                 })

       @WebMethod(operationName = "GetDeviceEnabled",
                           action = "http://www.nrf-
arts.org/UnifiedPOS/Belt/GetDeviceEnabled")

       @WebResult(name = "GetDeviceEnabledResult",
                  targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")

       @RequestWrapper(localName = "GetDeviceEnabled",
                       targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                       className = "org.arts.unifiedpos.belt.GetDeviceEnabled")

       @ResponseWrapper(localName = "GetDeviceEnabledResponse",
                       targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                       className = "org.arts.unifiedpos.belt.GetDeviceEnabledResponse")

       public Boolean getDeviceEnabled()
         throws FaultException;

       @Action(input = "http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabled",
              output = "http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabledResponse",
                fault = {@FaultAction(className =
org.arts.unifiedpos.belt.FaultException.class,
                         value = "http://www.nrf-arts.org/UnifiedPOS/Belt/UposException")
                        })

       @WebMethod(operationName = "SetDeviceEnabled",
                           action = "http://www.nrf-
arts.org/UnifiedPOS/Belt/SetDeviceEnabled")

       @RequestWrapper(localName = "SetDeviceEnabled",
                  targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                        className = "org.arts.unifiedpos.belt.SetDeviceEnabled")

       @ResponseWrapper(localName = "SetDeviceEnabledResponse",
                  targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                         className = "org.arts.unifiedpos.belt.SetDeviceEnabledResponse")
```

```
        public void setDeviceEnabled(
                    @WebParam(name = "DeviceEnabled",
                   targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")

                Boolean deviceEnabled)throws FaultException;

                        ...

         @Action(input="http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealth",
                output="http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealthResponse",
                 Fault={@FaultAction(className=org.arts.unifiedpos.belt.FaultException.class,
                          value="http://www.nrf-arts.org/UnifiedPOS/Belt/UposException")
                        })

         @WebMethod(operationName = "CheckHealth",
                           action = "http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealth")

         @RequestWrapper(localName = "CheckHealth",
                   targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                         className = "org.arts.unifiedpos.belt.CheckHealth")

         @ResponseWrapper(localName = "CheckHealthResponse",
                    targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                          className = "org.arts.unifiedpos.belt.CheckHealthResponse")
         public void checkHealth(
                    @WebParam(name = "Level",
                   targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")
                   HealthCheckLevel level)
                   throws FaultException;

                        ...
    }
/*--------- Belt.java end -------------*/
```

Because JAX-WS has no possibility to declare a DataContract as in C#, for WSDL input and output actions, separate classes are generated providing the XML data access:

/*--------- CheckHealth.java start ----------*/

```
package org.arts.unifiedpos.belt;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;



@XmlAccessorType(XmlAccessType.FIELD)

@XmlType(name = "", propOrder = { "level" })

@XmlRootElement(name = "CheckHealth")

public class CheckHealth
   {

       @XmlElement(name = "Level")
       protected HealthCheckLevel level;

       public HealthCheckLevel getLevel()
        {
           return level;
        }

       public void setLevel(HealthCheckLevel value)
        {
           this.level = value;
        }
  }
```

/*--------- CheckHealth.java end ----------*/

However, as the CheckHealth method does not return any data, the CheckHealthResponse class is empty:

/*--------- CheckHealthResponse.java start ----------*/

```
package org.arts.unifiedpos.belt;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;


@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "")
@XmlRootElement(name = "CheckHealthResponse")
public class CheckHealthResponse
   {
   }
```

/*--------- CheckHealthResponse.java end ----------*/

/*-- Example End*/