HITACHI
Inspire the Next

OpenTP1 Version 7
分散トランザクション処理機能

OpenTP1 プログラム作成の手引

手引書

3000-3-D51-90

前書き

■ 対象製品

マニュアル「OpenTP1 解説」を参照してください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

HITACHI，JP1，OpenTP1，OSAS，uCosminexus，XMAP は，株式会社 日立製作所の商標または登録商標です。

Oracle(R)，Java 及び MySQL は，Oracle，その子会社及び関連会社の米国及びその他の国における登録商標です。

UNIX は，The Open Group の登録商標です。

Windows は，マイクロソフト 企業グループの商標です。

その他記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

本書には，X/Open の許諾に基づき X/Open CAE Specification System Interfaces and Headers, Issue4,（C202 ISBN 1-872630-47-2）Copyright (C) July 1992，X/Open Company Limited の内容が含まれています；

なお，その一部は IEEE Std 1003.1-1990，(C) 1990 Institute of Electrical and Electronics Engineers, Inc.及び IEEE std 1003.2/D12，(C) 1992 Institute of Electrical and Electronics Engineers, Inc.を基にしています。

事前に著作権所有者の許諾を得ずに，本書の該当部分を複製，複写及び転記することは禁じられています。

本書には，X/Open の許諾に基づき X/Open Preliminary Specification Distributed Transaction Processing：The TxRPC Specification（P305 ISBN 1-85912-000-8）Copyright (C) July 1993, X/Open Company Limited の内容が含まれています；

事前に著作権所有者の許諾を得ずに，本書の該当部分を複製，複写及び転記することは禁じられています。

本書には，Open Software Foundation, Inc.が著作権を有する内容が含まれています。

This document and the software described herein are furnished under a license, and may be used and copied only in accordance with the terms of such license and with the inclusion of the above copyright notice. Title to and ownership of the document and software remain with OSF or its licensors.

■ 発行

2023 年 7 月　3000-3-D51-90

■ 著作権

All Rights Reserved. Copyright (C) 2006, 2023, Hitachi, Ltd.

変更内容

変更内容（3000-3-D51-90） uCosminexus TP1/Server Base 07-60，uCosminexus TP1/Server Base(64) 07-60

追加・変更内容	変更個所
追加，変更個所はない。	－

単なる誤字・脱字などはお断りなく訂正しました。

変更内容（3000-3-D51-80） uCosminexus TP1/Server Base 07-57，uCosminexus TP1/Server Base(64) 07-57，uCosminexus TP1/Server Base 07-56，uCosminexus TP1/Server Base(64) 07-56，uCosminexus TP1/Server Base 07-54，uCosminexus TP1/Server Base(64) 07-54，uCosminexus TP1/Server Base 07-53，uCosminexus TP1/Server Base(64) 07-53

追加・変更内容
マニュアル訂正の内容を反映した。
TAM サービスの関数とフラグに設定した値について説明を追加した。

変更内容（3000-3-D51-73） uCosminexus TP1/Server Base 07-54，uCosminexus TP1/Server Base(64) 07-54，uCosminexus TP1/Server Base 07-53，uCosminexus TP1/Server Base(64) 07-53

追加・変更内容
エントリポイント名に関する説明を追加した。
セクタ長に関する説明を追加した。
環境変数の設定例を追加した。

変更内容（3000-3-D51-72） uCosminexus TP1/Server Base 07-53，uCosminexus TP1/Server Base(64) 07-53

追加・変更内容
コーディング時の注意事項に記載しているマルチスレッドに関する説明を変更した。
マルチ OpenTP1 のコマンドを振り分けるサンプルに関する説明を変更した。

変更内容（3000-3-D51-71） uCosminexus TP1/Server Base 07-51，uCosminexus TP1/Server Base(64) 07-51，uCosminexus TP1/Message Control 07-51，uCosminexus TP1/Message

Control(64) 07-51，uCosminexus TP1/NET/Library 07-51，uCosminexus TP1/NET/Library(64) 07-51

追加・変更内容
マニュアル訂正の内容を反映した。

変更内容（3000-3-D51-70） uCosminexus TP1/Server Base 07-51，uCosminexus TP1/Server Base(64) 07-51，uCosminexus TP1/Message Control 07-51，uCosminexus TP1/Message Control(64) 07-51，uCosminexus TP1/NET/Library 07-51，uCosminexus TP1/NET/Library(64) 07-51

追加・変更内容
ERREVT2 が通知された原因に関する説明を追加した。
次の時間監視の設定時間と所要時間の誤差に関する説明を追加した。 • 同期型のメッセージ処理の時間監視 • アプリケーションプログラムの起動のタイマ起動 • 非トランザクション属性の MHP の時間監視 • ユーザタイマ監視機能による時間監視
make コマンドを実行するタイミングについての説明を追加した。

変更内容（3000-3-D51-60） uCosminexus TP1/Server Base 07-50，uCosminexus TP1/Server Base(64) 07-50，uCosminexus TP1/Message Control 07-50，uCosminexus TP1/Message Control(64) 07-50，uCosminexus TP1/NET/Library 07-50，uCosminexus TP1/NET/Library(64) 07-50

追加・変更内容
非常駐プロセスを終了させる場合の，スケジュールキューに滞留しているサービス要求の数について，説明を変更した。
dc_mcf_execap 関数でアプリケーションを起動する場合に関連づけるシステム定義に関する説明を追加した。
閉塞されている論理端末のメッセージが未処理送信メッセージ滞留時間の対象とならないことを追加した。
排他なし参照使用時の注意事項を追加した。

変更内容（3000-3-D51-50） uCosminexus TP1/Server Base 07-06，uCosminexus TP1/Server Base(64) 07-06

追加・変更内容
一時的にプロセス数が増加するタイミング，および非常駐プロセスでも同一プロセスで続けてサービス要求を処理する場合について，説明を追加した。
性能検証用トレースの情報を CSV 形式で出力し，トレース解析できるようにした。
ノード構成の変更（ノードの追加や削除）に自動的に対応する機能（ノード自動追加機能）を追加した。

追加・変更内容
再送できるメッセージの条件と，システムサービス定義との関連について説明を追加した。

uCosminexus TP1/Server Base 07-05，uCosminexus TP1/Server Base(64) 07-05，uCosminexus TP1/Message Control 07-05，uCosminexus TP1/Message Control(64) 07-05

追加・変更内容
一つのサービス要求ごとに実行するプロセスを起動し直せるようにした（非常駐 UAP プロセスのリフレッシュ機能）。
一つのリソースマネジャを複数の制御単位に分け，接続するユーザ名称などを変更してリソースマネジャに接続できるようにした（リソースマネジャ接続先選択機能）。

uCosminexus TP1/Message Control 07-00，uCosminexus TP1/Message Control(64) 07-00

追加・変更内容
リモート MCF サービスに関連する記述を削除した。

変更内容（3000-3-D51-40） uCosminexus TP1/Server Base 07-04，uCosminexus TP1/Server Base(64) 07-04，uCosminexus TP1/Message Control 07-05，uCosminexus TP1/Message Control(64) 07-05，uCosminexus TP1/NET/Library 07-05，uCosminexus TP1/NET/Library(64) 07-05

追加・変更内容
サービス関数動的ローディング機能で使用する，UAP 共用ライブラリのサーチパスをオンライン中に変更できるようにした。
OpenTP1 の起動コマンドがリターンした直後に MCF の運用コマンドを実行する場合，mcftlscom コマンドで MCF 通信サービスの開始を待ち合わせられるようにした。
mcftlsle コマンドで，最大未送信メッセージ数を表示できるようにした。 これに伴い，dc_mcf_tlsle 関数との機能差異の説明を追加した。
非応答型の MHP からの問い合わせ応答をできるようにした。
異常終了した MHP を，自動的に再スケジュールできるようにした。
TX_関数の使用方法で，ユーザサービス定義との関連について説明を変更した。

変更内容（3000-3-D51-30） uCosminexus TP1/Server Base 07-03，uCosminexus TP1/Server Base(64) 07-03，uCosminexus TP1/Message Control 07-03，uCosminexus TP1/Message Control(64) 07-03，uCosminexus TP1/NET/Library 07-04，uCosminexus TP1/NET/Library(64) 07-04

追加・変更内容
OpenTP1 の UAP をコーディングする場合の注意事項を追加した。
通知する先の CUP の説明に，dc_clt_chained_accept_notification 関数を追加した。

追加・変更内容
グローバルドメインについての説明を追加した。
OpenTP1 の標準出力，標準エラー出力をリダイレクトする prctee プロセスを停止・再開始できるようにした。 これに伴い，次のコマンドを追加した。 • prctctrl
アプリケーションに関するタイマ起動要求の状態を表示できるようにした。 これに伴い，次のコマンドを追加した。 • mcfalstap
ユーザタイマ監視の状態を表示できるようにした。 これに伴い，次のコマンドを追加した。 • mcftlsutm
UAP 異常終了通知イベント，および未処理送信メッセージ廃棄通知イベントの，MCF イベントが通知された原因の説明を変更した。
OpenTP1 の正常終了コマンドを実行すると，タイマ起動要求メッセージはすぐに破棄され，ERREVTA が通知される旨の説明を追加した。
サンプルディレクトリの，tools/ディレクトリの説明を変更した。

uCosminexus TP1/Message Control 07-02，uCosminexus TP1/NET/Library 07-03

追加・変更内容
MHP でサービス関数動的ローディング機能を使用できるようにした。
MCF 通信サービスまたはアプリケーション起動サービスの状態を，ライブラリ関数で表示できるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tlscom • CBLDCMCF('TLSCOM ')
コネクションの状態表示，確立，および解放を，ライブラリ関数でできるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tactcn • dc_mcf_tdctcn • dc_mcf_tlscn • CBLDCMCF('TACTCN ') • CBLDCMCF('TDCTCN ') • CBLDCMCF('TLSCN ')
サーバ型コネクションの確立要求の受付開始・終了を，ライブラリ関数でできるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tofln • dc_mcf_tonln • CBLDCMCF('TOFLN ')

追加・変更内容
• CBLDCMCF('TONLN ')
コネクションの確立要求の受付状態を，ライブラリ関数で表示できるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tlsln • CBLDCMCF('TLSLN ')
アプリケーションに関するタイマ起動要求を，ライブラリ関数で削除できるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_adltap • CBLDCMCF('ADLTAP ')
論理端末の状態表示，閉塞，閉塞解除，および出力キューの削除を，ライブラリ関数でできるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tactle • dc_mcf_tdctle • dc_mcf_tdlqle • dc_mcf_tlsle • CBLDCMCF('TACTLE ') • CBLDCMCF('TDCTLE ') • CBLDCMCF('TDLQLE ') • CBLDCMCF('TLSLE ')
相手システムとのメッセージ送受信に関するネットワークの状態を表示できるようにした。 これに伴い，次のコマンドを追加した。 • mcftlsln
運用操作で使用する関数と運用コマンドの機能差異についての説明を追加した。
通信プロトコル対応製品と運用操作で使用できる関数の対応についての説明を追加した。

uCosminexus TP1/Message Control 07-01，uCosminexus TP1/NET/Library 07-01

追加・変更内容
サーバ型コネクションの確立要求の受付開始・終了を，手動でできるようにした。 これに伴い，次のコマンドを追加した。 • mcftofln • mcftonln
リアルタイム統計情報の取得項目として，MCF の情報も取得できるようにした。

変更内容（3000-3-D51-20） uCosminexus TP1/Server Base 07-02，uCosminexus TP1/Message Control 07-01，uCosminexus TP1/NET/Library 07-01

追加・変更内容
監査ログを出力する機能を追加した。 これに伴い，UAP で監査ログを取得する実装方法を追加した。
サービス関数を動的にローディングできる機能を追加した。
リモート API 機能に関する説明を変更した。

はじめに

このマニュアルは，次に示す OpenTP1 のプログラムプロダクトで使うアプリケーションプログラムの作成方法について説明しています。

- 分散トランザクション処理機能　TP1/Server Base
- 分散アプリケーションサーバ　TP1/LiNK

このマニュアルでは，アプリケーションプログラムの英略称を「ユーザが作成するアプリケーションプログラム」の意味で，UAP（User Application Program）と表記します。

本文中に記載されている製品のうち，このマニュアルの対象製品ではない製品については，OpenTP1 Version 7 対応製品の発行時期をご確認ください。

次に示す製品，および各製品に示したバージョン以降で，ソケット受信型サーバに関する機能はすべて廃止しました。そのため，ユーザサービス定義とユーザサービスデフォルト定義の receive_from オペランドで socket は使用できません。

- P-1M64-2141 uCosminexus TP1/Server Base：07-53-01 以降
- P-1M64-1121 uCosminexus TP1/Server Base(64)：07-53-01 以降
- P-1J64-2171 uCosminexus TP1/Server Base：07-51-02 以降
- P-1J64-1171 uCosminexus TP1/Server Base(64)：07-51-01 以降
- P-8164-2111 uCosminexus TP1/Server Base：07-57 以降
- P-8264-2111 uCosminexus TP1/Server Base(64)：07-57 以降
- P-2464-2294 uCosminexus TP1/Server Base：07-60 以降
- P-2964-2234 uCosminexus TP1/Server Base(64)：07-60 以降

なお，該当する機能を使用した場合の動作は保証できないため，ご注意ください。

■ 対象読者

TP1/Server Base，または TP1/LiNK で使うアプリケーションプログラムを作成するプログラマの方々を対象としています。

オペレーティングシステム，オンラインシステム，使うマシンの操作，およびアプリケーションプログラムのコーディングに使う高級言語（C 言語，C++言語，または COBOL 言語）の文法の知識があることを前提としています。

このマニュアルの記述は，マニュアル「OpenTP1 解説」，またはマニュアル「TP1/LiNK 使用の手引」の知識があることを前提としていますので，あらかじめお読みいただくことをお勧めします。

■ 文法の記号

このマニュアルで使用する各種記号を説明します。

(1)文法記述記号

指定する値の説明で使用する記号の一覧を示します。

文法記述記号	意味
【　】 隅付き括弧	C 言語の関数名に対応する COBOL 言語の関数名をこの記号で囲んで表記しています。それ以降は，C 言語の関数名に統一して説明します。

■ X/Open 発行のドキュメントの内容から引用した記述について

X/Open 発行の「X/Open CAE Specification Distributed Transaction Processing : The XATMI Specification」の内容から引用した部分

上記ドキュメントに示された仕様の解釈を，OpenTP1 での使用方法として，このマニュアルの次に示す部分に記載しました。

- 5 章　X/Open に準拠したアプリケーションプログラミングインタフェース

5.1　XATMI インタフェース（クライアント／サーバ形態の通信）

X/Open 発行の「X/Open CAE Specification Distributed Transaction Processing : The TX（Transaction Demarcation） Specification」の内容から引用した部分

上記ドキュメントに示された仕様の解釈を，OpenTP1 での使用方法として，このマニュアルの次に示す部分に記載しました。

- 5 章　X/Open に準拠したアプリケーションプログラミングインタフェース

5.2　TX インタフェース（トランザクション制御）

X/Open 発行の「X/Open Preliminary Specification Distributed Transaction Processing : The TxRPC Specification」の内容から引用した部分

上記ドキュメントに示された仕様の解釈を，OpenTP1 での使用方法として，このマニュアルの次に示す部分に記載しました。

- 6 章　X/Open に準拠したアプリケーション間通信（TxRPC）

■ 謝　辞

COBOL 言語仕様は，CODASYL（the Conference on Data Systems Languages：データシステムズ言語協議会）によって，開発された。OpenTP1 のアプリケーションプログラムのインタフェース仕様のうち，データ操作言語（DML　Data Manipulation Language）の仕様は，CODASYL　COBOL（1981）の通信節，RECEIVE 文，SEND 文，COMMIT 文，及び ROLLBACK 文を参考にし，それに日立製作所独自の解釈と仕様を追加して開発した。原開発者に対し謝意を表すとともに，CODASYL の要求に従って以下の謝辞を掲げる。なお，この文章は，COBOL の原仕様書「CODASYL COBOL JOURNAL OF DEVELOPMENT 1984」の謝辞の一部を再掲するものである。

いかなる組織であっても，COBOL の原仕様書とその仕様の全体又は一部分を複製すること，マニュアルその他の資料のための土台として原仕様書のアイデアを利用することは自由である。ただし，その場合には，その刊行物のまえがきの一部として，次の謝辞を掲載しなければならない。書評などに短い文章を引用するときは，"COBOL"という名称を示せば謝辞全体を掲載する必要はない。

COBOL は産業界の言語であり，特定の団体や組織の所有物ではない。

CODASYL　COBOL 委員会又は仕様変更の提案者は，このプログラミングシステムと言語の正確さや機能について，いかなる保証も与えない。さらに，それに関連する責任も負わない。

次に示す著作権表示付資料の著作者及び著作権者

FLOW-MATIC（Sperry Rand Corporation の商標），Programming for the Univac

（R）I and II，Data Automation Systems，　Sperry Rand Corporation　著作権表示

1958 年，1959 年；

IBM Commercial Translator Form No.F 28-8013，IBM 著作権表示 1959 年;

FACT，DSI 27A5260-2760，Minneapolis-Honeywell，著作権表示 1960 年

は，これら全体又は一部分を COBOL の原仕様書中に利用することを許可した。この許可は，COBOL 原仕様書をプログラミングマニュアルや類似の刊行物に複製したり，利用したりする場合にまで拡張される。

■ KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

■ その他の前提条件

このマニュアルをお読みになる際のその他の前提情報については，マニュアル「OpenTP1 解説」を参照してください。

目次

1 OpenTP1 のアプリケーションプログラム

OpenTP1 のアプリケーションプログラムの概要について説明します。

この章では，各機能を **C 言語の関数名**で説明します。C 言語の関数名に対応する COBOL 言語の API は，関数を最初に説明する個所に【 】で囲んで表記します。それ以降は，C 言語の関数名に統一して説明します。

1.1　ユーザアプリケーションプログラムと業務形態の関係

OpenTP1※のアプリケーションプログラム（UAP　User Application Program）は，ネットワーク（LAN，または WAN）でつながれているメインフレーム，ワークステーション（WS），パーソナルコンピュータ（PC），および分散機と通信する，オンライントランザクション処理を実現するために作成します。

OpenTP1 の UAP でできる通信形態は，次の三つです。

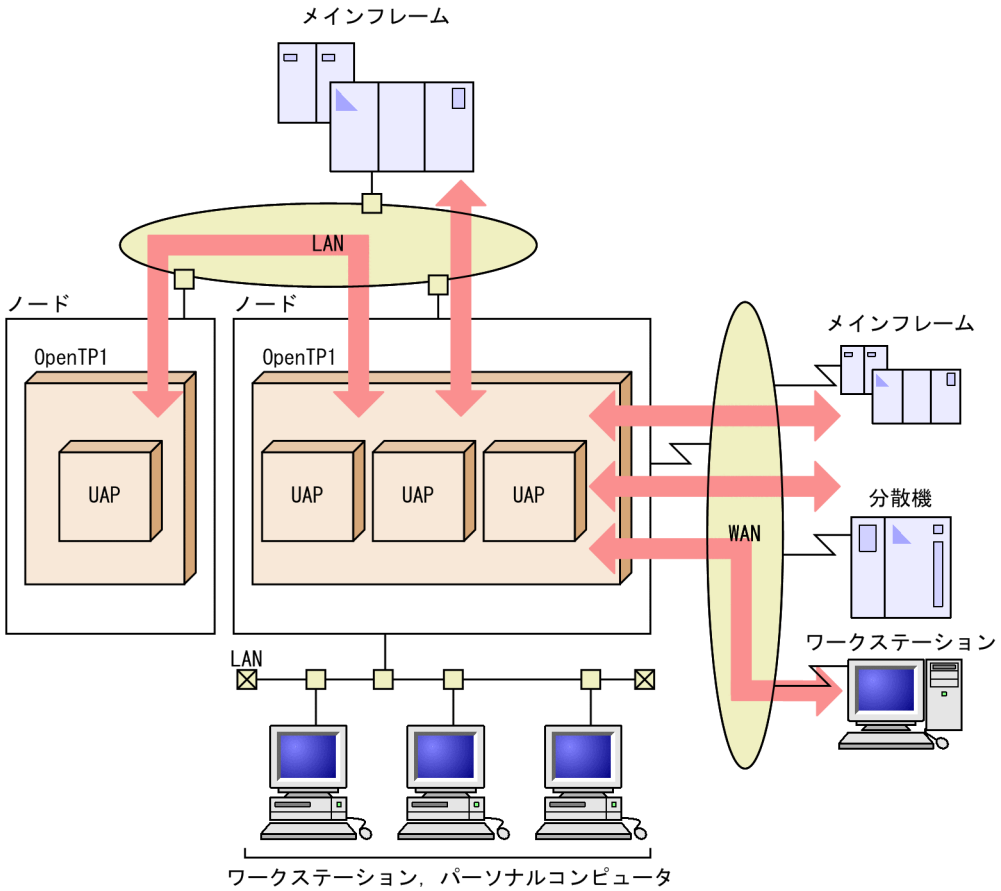
- クライアント／サーバ形態の UAP
- メッセージ送受信形態の UAP
- メッセージキューイング機能を使った形態の UAP

注※

このマニュアルでは，分散トランザクション処理機能 TP1/Server Base と分散アプリケーションサーバ TP1/LiNK を総称して，以降 OpenTP1 と表記します。

OpenTP1 と UAP のネットワーク内の位置を次の図に示します。

図 1-1　OpenTP1 と UAP のネットワーク内の位置



1.1.1　クライアント／サーバ形態のアプリケーションプログラム

クライアント／サーバ形態の UAP では，ほかの UAP の処理を呼び出して業務処理を実行できます。呼び出して利用できるプログラムの単位を**サービス**，サービスを提供するプロセスを**サーバ**といいます。UAP のサーバを**ユーザサーバ**といいます。

クライアント／サーバ形態の通信では，サービスを要求する UAP（**クライアント UAP**）とサービスを提供する UAP（**サーバ UAP**）で一つの業務になります。サーバ UAP のサービスは，複数のクライアント UAP で共用できます。

サービスを要求するときは，**リモートプロシジャコール（RPC）**を使います。RPC では，サーバ UAP に論理的に付けた名称（**サービス名**）でサービスを要求できます。このときクライアント UAP では，サーバ UAP がネットワーク上のどのノードにあるかを意識する必要はありません。

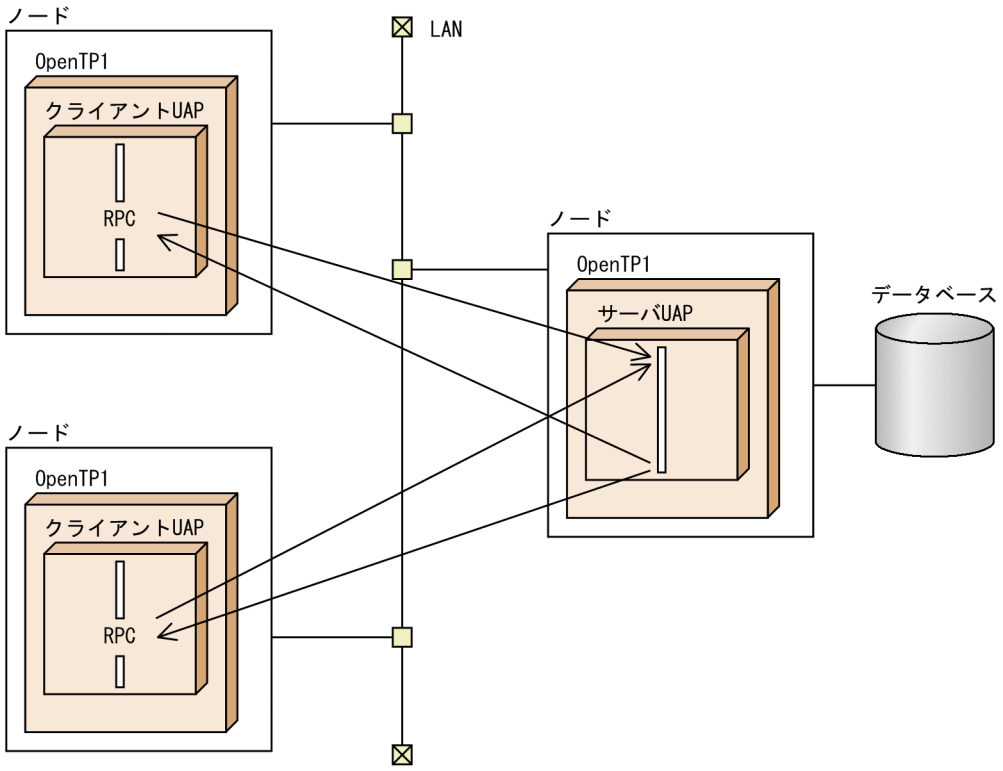
OpenTP1 では，クライアント／サーバ形態の通信プロトコルに TCP/IP と OSI TP を使えます。どちらを使う場合でも，UAP でノード間の通信プロトコルを意識する必要はありません。

ノードについて

このマニュアルで意味する「ノード」とは，ネットワークにつながれた，OpenTP1 が稼働する一つの計算機（マシン）のことです。ただし，マルチ OpenTP1 の場合は，複数の OpenTP1 から構成される一つのノードになります。

クライアント／サーバ形態の UAP の概要を次の図に示します。

図 1-2　クライアント／サーバ形態の UAP の概要



1.1.2　メッセージ送受信形態のアプリケーションプログラム

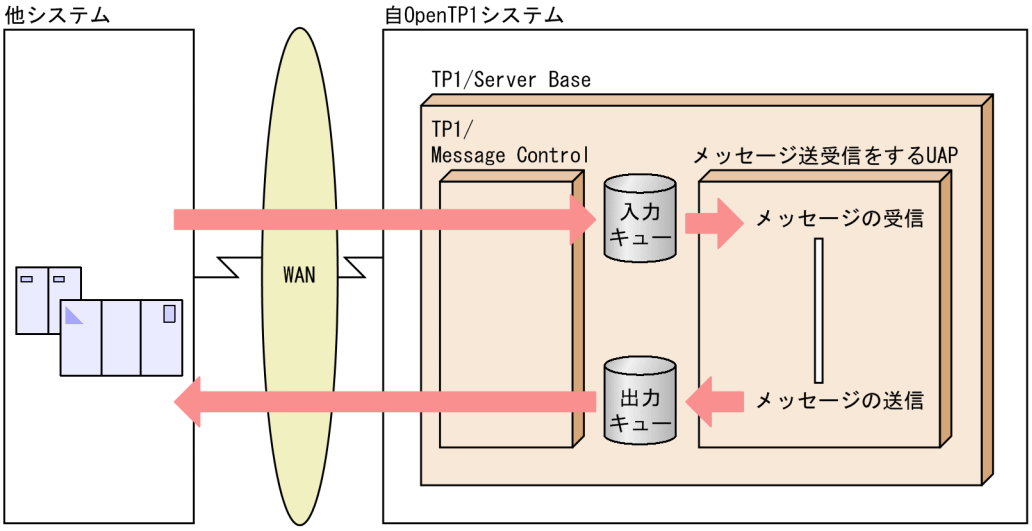
メッセージ送受信形態の UAP では，OSI に準拠したプロトコル（OSAS/UA，OSI TP），TCP/IP，および 従来型のネットワークで接続されたホストと分散機間で，メッセージの送受信による通信ができます。主に，通信管理プログラムを介した広域ネットワーク（WAN）にある他システムとの通信で使います。

メッセージ送受信をする UAP とクライアント／サーバ形態の RPC を使う UAP では，コーディングスタイルが異なります。クライアント／サーバ形態の処理で使う UAP とは別に，メッセージ送受信専用の UAP として作成します。

メッセージ送受信形態の UAP を使うノードには，OpenTP1 の**メッセージ送受信機能**（TP1/Message Control，TP1/NET/Library）と**通信プロトコル対応製品**（TP1/NET/***）が必要です。このマニュアルでは，OpenTP1 のメッセージ送受信機能（TP1/Message Control，TP1/NET/Library）と通信プロトコル対応製品を総称して，以降 **MCF**（Message Control Facility）または **MCF サービス**と表記します。

メッセージ送受信形態の UAP の概要を次の図に示します。

図 1-3　メッセージ送受信形態の UAP の概要



1.1.3　メッセージキューイング機能を使った形態のアプリケーションプログラム

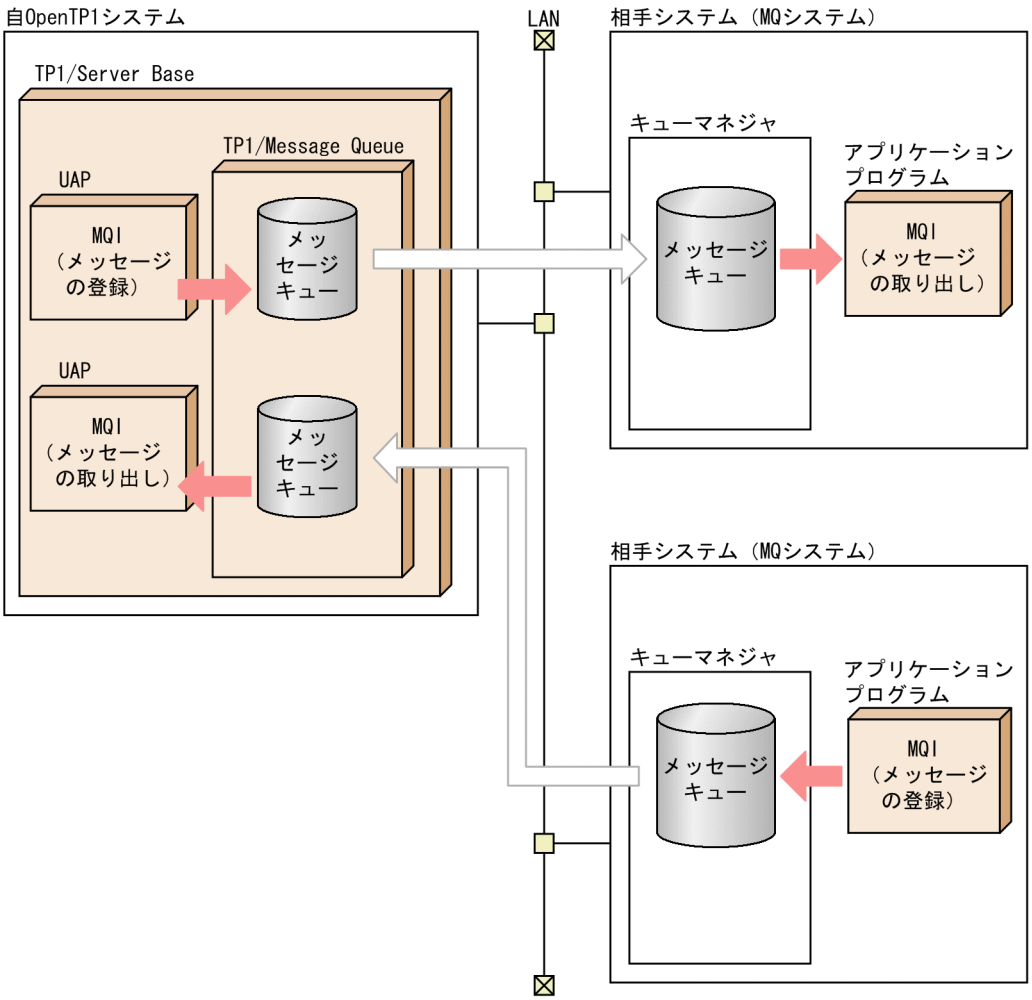
メッセージキューイング機能とは，データを格納するキュー（**メッセージキュー**）に情報を登録したり，情報を取り出したりして通信する形態です。相手システムのアプリケーションプログラムが稼働していなくても，データを送信したり受信したりできるので，電子メールのように通信できます。

メッセージキューイング機能を使うときは，UAP から**メッセージキューインタフェース（MQI）**という API を使って操作します。

メッセージキューイング機能を使う OpenTP1 のノードには，TP1/Message Queue が必要です。メッセージキューイング機能の使い方については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

メッセージキューイング機能を使った UAP の概要を次の図に示します。

図 1-4　メッセージキューイング機能を使った UAP の概要



1.1.4　アプリケーションプログラムの負荷分散

OpenTP1 では，UAP を複数のプロセスで稼働させることで，効率良く業務を実行できます。ノード内では，一つの UAP の処理を複数のプロセスで実行させて，サーバシステムの効率を上げています。この機能を**マルチサーバ**といいます。また，同じ名称の UAP を複数のマシンに存在させて，サービス要求をどのノードでも処理できるようにもできます。この機能を**ノード間負荷バランス機能**といいます。UAP と実行するプロセスの関係については，「1.3.6　ユーザサーバの負荷分散とスケジュール」を参照してください。

1.1.5　アプリケーションプログラムのトランザクション処理

UAP の処理は，業務処理ごとの単位に区切って，それぞれの処理の結果を有効にするか無効にするかを明確に分ける必要があります。処理が有効であるか無効であるかどちらかに必ず決定させる単位を**トランザクション**といいます。OpenTP1 の UAP では，このようなトランザクションの処理ができます。

トランザクションの業務処理ごとの区切りを**同期点**といいます。トランザクションの処理が同期点に達した時点で，トランザクションの処理が正常に終了したか（有効）異常が起こったか（無効）を決定します。処理が正常に終了したとする同期点取得を**コミット**といいます。同期点まで正常に終了できなかったトランザクションの処理は，OpenTP1 でそれまでの処理を取り消して，その処理がなかったように回復します。このような同期点処理を**ロールバック**（部分回復）といいます。

(1)　クライアント／サーバ形態の UAP でのトランザクション処理

OpenTP1 では，RPC を使ったクライアント／サーバ形態の UAP の処理をトランザクションとして処理できます。異なるノードにわたって，多くのサービスを続けて要求している処理でも，一つのトランザクションの処理にできます。

クライアント／サーバ形態の UAP では，**トランザクションの開始**と，**同期点取得**を示す関数を呼び出せます。トランザクションの開始を宣言した UAP から複数のサービスをネストさせても，一つのトランザクションとして処理できます。

このように OpenTP1 では，従来のデータコミュニケーションでのトランザクション処理の信頼性を，クライアント／サーバ形態の UAP で実現できます。

(2)　メッセージ送受信形態のトランザクション処理

メッセージを処理する UAP の処理は，開始から終了まで，トランザクションにできます。この場合の同期点処理は OpenTP1 で自動的に制御しています。

メッセージを処理する UAP でメッセージを受け取ったあとでは，クライアント／サーバ形態の UAP で使うトランザクション制御の関数は使えません。

1.2 アプリケーションプログラムの種類

OpenTP1 の UAP には，次に示す種類があります。

- クライアント／サーバ形態の通信で使う UAP
  - サービス利用プログラム（SUP　Service Using Program）

    クライアント専用の UAP です。SUP は，OpenTP1 の基本機能（TP1/Server Base，または TP1/LiNK）が前提となります。
  - サービス提供プログラム（SPP　Service Providing Program）

    クライアント UAP からの要求に対して，サービスを提供する UAP（サーバ UAP）です。SPP は，OpenTP1 の基本機能（TP1/Server Base，または TP1/LiNK）が前提となります。
- メッセージ送受信形態の通信で使う UAP
  - メッセージ処理プログラム（MHP　Message Handling Program）

    通信回線を経由して送られたメッセージを受信して処理する UAP です。MHP の処理から SPP のサービスも要求できます。MHP は，OpenTP1 の基本機能（TP1/Server Base）とメッセージ送受信機能（TP1/Message Control）が前提となります。
- ユーザファイルの初期化をする UAP
  - オフラインの業務をする UAP

    ユーザ任意の処理をする UAP です。オフラインの業務をする UAP で使える OpenTP1 のライブラリ関数は，DAM ファイルを初期作成したり，バッチ環境でアクセスしたりする機能だけです。
- OpenTP1 クライアント機能（TP1/Client）で使う UAP
  - クライアントユーザプログラム（CUP　Client User Program）

    クライアント専用の UAP です。WS，または PC から TP1/Client のライブラリ関数を使って，SPP のサービスを要求するプログラムを総称して CUP といいます。CUP は，OpenTP1 クライアント機能（TP1/Client）が前提となります。

    CUP については，マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」を参照してください。

    なお，TP1/Client/J を使用すると，SPP のサービスを要求する Java アプレット，Java アプリケーション，および Java サーブレットを作成できます。

    詳細については，マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/J 編」を参照してください。

クライアント／サーバ形態の UAP（SUP，SPP）の概要を図 1-5 に，メッセージ送受信形態の UAP（MHP）の概要を図 1-6 に示します。

図 1-5　UAP の役割と位置の概要（クライアント／サーバの形態）

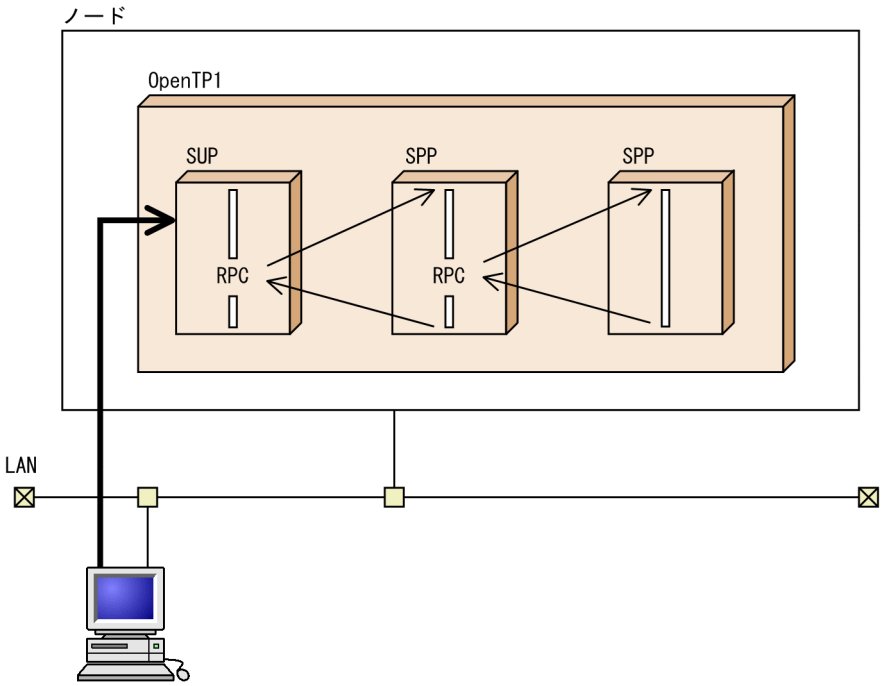
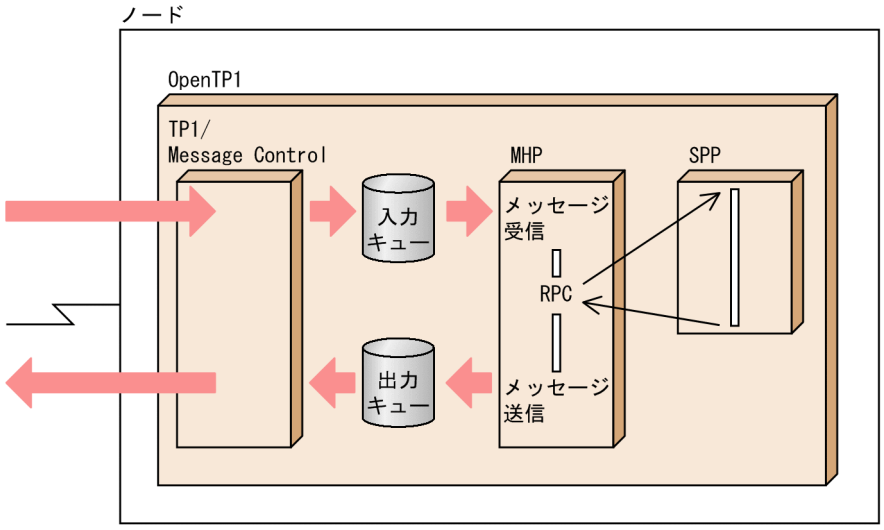


図 1-6　UAP の役割と位置の概要（メッセージ送受信の形態）



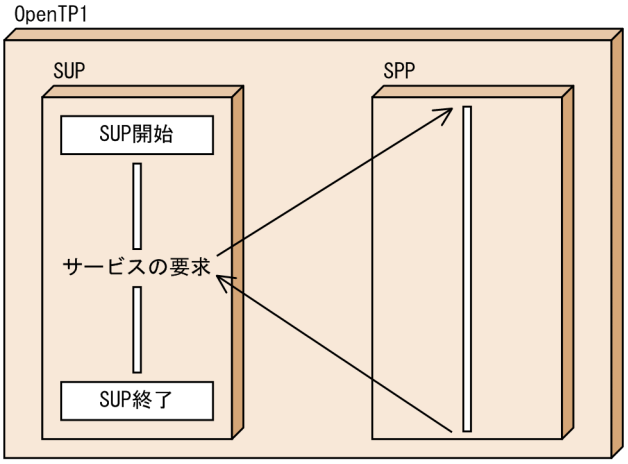
1.2.1　サービスを利用する UAP（SUP）

クライアント専用の UAP を，**サービス利用プログラム（SUP）**といいます。SUP は，サーバ UAP（SPP）にサービスを要求して，クライアント／サーバ形態の通信を開始する役割の UAP です。

SUP でできる通信は，SPP にサービスを要求するだけです。ほかの UAP にサービスとして提供するための関数は作成できません。

SUP の概要を次の図に示します。

図 1-7　SUP の概要



(1)　SUP の開始

SUP を実行する場合，**OpenTP1 の開始と一緒に開始する方法**と，**OpenTP1 の開始後に任意に開始する方法**の 2 とおりがあります。OpenTP1 の開始と一緒に開始すると，OpenTP1 の開始と同時に UAP の業務を開始できます。作成した SUP の業務内容に応じて，開始する時期を選べます。

(a)　OpenTP1 の開始と一緒に開始する場合

OpenTP1 を開始する前に，OpenTP1 と一緒に開始する指定をしておきます。指定方法を次に示します。

- TP1/Server Base の場合
  ユーザサービス構成定義の **dcsvstart 定義コマンド**に，開始する SUP のユーザサーバ名を指定します。
- TP1/LiNK の場合
  ユーザサーバ環境を設定するときに，開始する SUP が自動起動するように設定します。

(b)　OpenTP1 の開始後に任意に開始する場合

OpenTP1 の開始後に SUP を開始する場合は，**dcsvstart コマンド**の引数に **SUP のユーザサーバ名**を指定して実行します。

(2)　SUP の稼働時

SUP のプロセスは，一つの常駐プロセスとして確保しておきます。

オンライン中に SUP のプロセスで障害が起こった場合は，自動的に別プロセスで開始できます。別プロセスに自動的に開始させる場合，TP1/Server Base のときは，ユーザサービス定義の auto_restart オペランドに Y を指定してください。TP1/LiNK のときは，自動的に開始するように設定されています。

OpenTP1 で自動的に開始できない場合は，**dcsvstart コマンド**で開始させてください。

(3) SUP の終了

SUP の終了は OpenTP1 で制御しません。業務終了後に SUP を正常終了させる場合は，SUP 自身で終了するように作成してください。SUP の処理からトランザクションを開始しているときは，関数でトランザクションをコミット（同期点を取得）してから終了させてください。処理がうまくいかなかったため SUP を異常終了させたい場合は，exit()または abort()を使って，SUP 自身で終了するように作成してください。

SUP は，dcsvstop コマンドで正常終了させることはできません。ただし，SUP を強制停止させたい場合に限り，dcsvstop -f コマンドで終了できます。

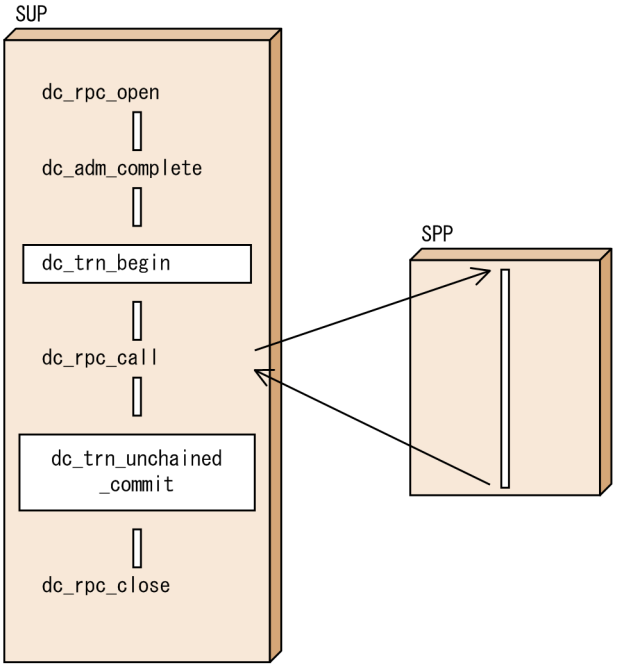
SUP のプロセスを，kill コマンドで終了させないでください。

(4) SUP の処理の概要

SUP では，UAP の開始（**dc_rpc_open 関数【CBLDCRPC('OPEN ')】**）を呼び出したあとに，サーバの起動完了を OpenTP1 に連絡するために，ユーザサーバの開始処理完了の報告（**dc_adm_complete 関数【CBLDCADM('COMPLETE')】**）を必ず呼び出してください。

SUP の処理の概要を次の図に示します。

図 1-8　SUP の処理の概要（C 言語の例）



1.2.2　サービスを提供する UAP（SPP）

要求されたサービスを処理する UAP を，**サービス提供プログラム（SPP）**といいます。SPP は OpenTP1 が稼働している間，クライアント UAP から要求されたサービスを処理します。クライアント UAP からは

関数呼び出しと同様の方法で，SPP のサービスを要求します。SPP がどのノードにあるかは，クライアント UAP で意識する必要はありません。

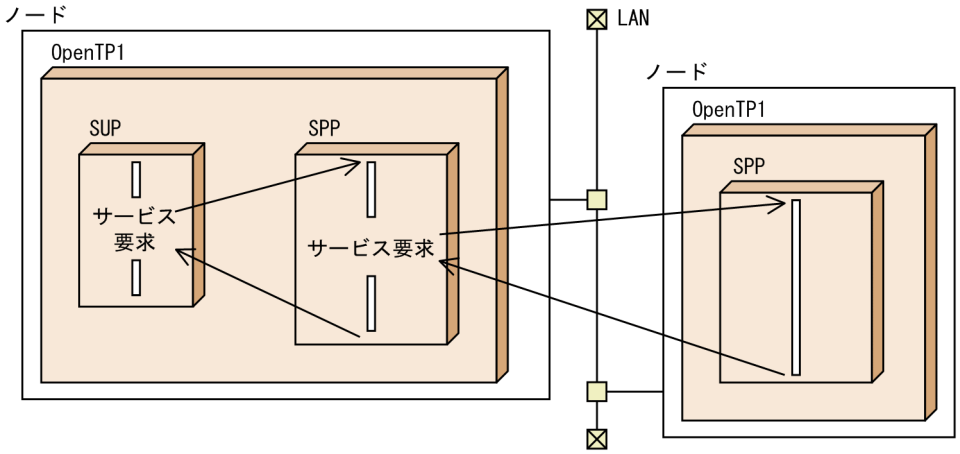
SPP は，サービスを要求されてから業務を開始します。サービスを要求されない間は，要求されるのを待っている状態となります。

SPP では，OpenTP1 のノードにあるユーザファイルへアクセスして，サーバの業務をします。OpenTP1 専用のファイルへライブラリ関数でアクセスしたり，ORACLE などの DBMS へ SQL 文でアクセスしたりできます。

SPP からさらに別の SPP へサービスを要求して，業務処理をネストさせることもできます。

SPP の概要を次の図に示します。

図 1-9　SPP の概要



(1)　SPP の構成

各種クライアント UAP の要求に対応するサービスを複数作成して，SPP として一つの実行形式ファイルにまとめます。一つ一つのサービスを，C 言語の場合は**サービス関数**（COBOL 言語の場合は**サービスプログラム**）といいます。SPP として一つの実行形式ファイルにするために，複数のサービスを**メイン関数**（COBOL 言語の場合は**メインプログラム**）でまとめます。そして，一つのメイン関数と複数のサービス関数から構成される SPP の実行形式ファイルを，サービスグループとして OpenTP1 に定義します。

サービス関数動的ローディング機能は，複数のサービスを UAP 共用ライブラリ化※して使うため，複数のサービスをメイン関数にまとめる作業は不要です。

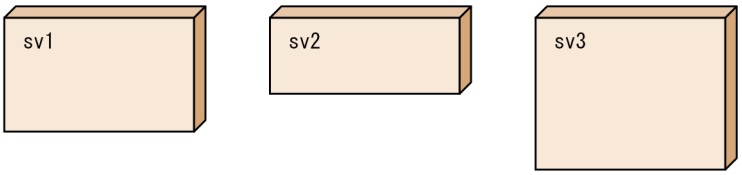
注※

UAP 共用ライブラリ化とは，UAP のソースファイルを翻訳（コンパイル）して作成した UAP オブジェクトファイルを結合（リンケージ）して，共用ライブラリとしてまとめることです。

SPP の構成を，スタブを使う場合とサービス関数動的ローディング機能を使う場合に分けて，それぞれ以降の図に示します。

図 1-10　SPP の構成（スタブを使う場合）

1. クライアントUAPに提供するサービスを，サービス関数として作成します。



2. メイン関数を作成して，翻訳，結合をすれば，SPPの実行形式ファイルとなります。

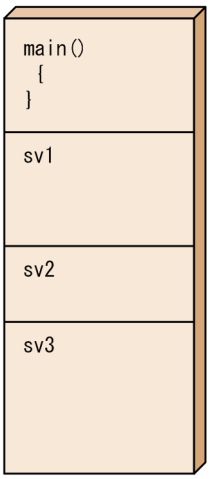
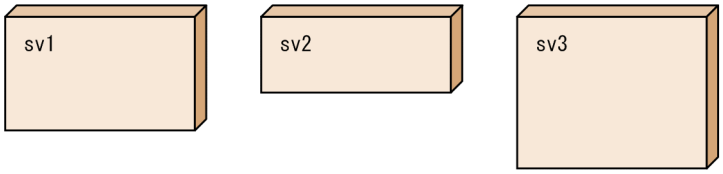
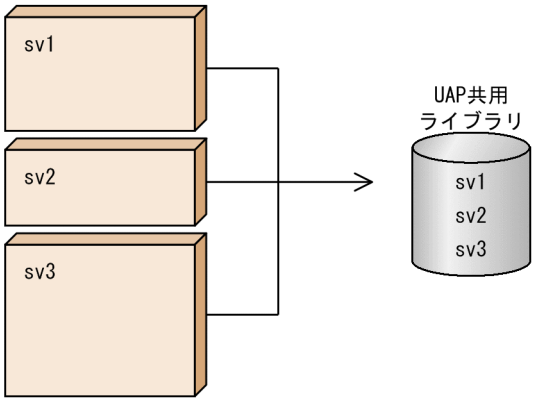


図 1-11　SPP の構成（サービス関数動的ローディング機能を使う場合）

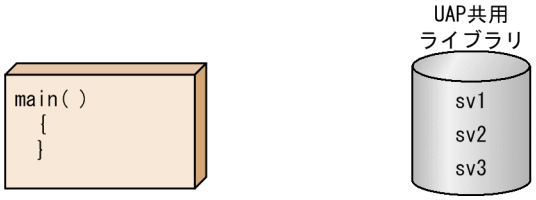
1. クライアントUAPに提供するサービスを，サービス関数として作成します。



2. サービス関数をUAP共用ライブラリとしてまとめます。



3. メイン関数を作成してから，翻訳してSPPの実行形式ファイルを作成します。
SPP実行時は，2. で作成したUAP共用ライブラリからSPP起動時にサービス関数を取得して実行します。



(2)　SPP の開始

SPP を実行する場合，**OpenTP1 の開始と一緒に開始する方法**と，**OpenTP1 の開始後に任意に開始する方法**の 2 とおりがあります。OpenTP1 の開始と一緒に開始すると，OpenTP1 の開始と同時に，SPP の業務を開始できます。SPP の業務内容に応じて，開始する時期を選べます。

(a)　OpenTP1 の開始と一緒に開始する場合

OpenTP1 を開始する前に，OpenTP1 と一緒に開始する指定をしておきます。指定方法を次に示します。

- TP1/Server Base の場合
  ユーザサービス構成定義の **dcsvstart 定義コマンド**に，開始する SPP のユーザサーバ名を指定します。
- TP1/LiNK の場合
  ユーザサーバ環境を設定する操作で，開始する SPP が自動起動するように設定します。

(b) OpenTP1 の開始後に任意に開始する場合

OpenTP1 の開始後に任意に開始する場合は，dcsvstart コマンドの引数に SPP のユーザサーバ名を指定して実行します。

SPP のプロセスはメイン関数から開始します。メイン関数で呼び出す，SPP のサービス開始の関数（dc_rpc_mainloop 関数【CBLDCRSV('MAINLOOP')】）が正常に実行されたことで，サービスを提供できる状態になります。

(3) SPP の稼働中

開始させた SPP は，メモリを効率的に使うため，事前に指定したプロセスの状態で稼働しています。開始させた SPP を**常駐プロセス**で稼働させる場合と，**非常駐プロセス**で稼働させる場合があります。常駐プロセスとした場合は，サービス要求が来ると SPP の処理を開始します。非常駐プロセスとしてある場合でも，サービス要求が来るとプロセスを自動的に起動して SPP の処理を開始します。

UAP プロセスに関する設定内容については，「1.3.5 アプリケーションプログラムの環境設定」を参照してください。

(4) SPP の終了

SPP が正常終了するのは，次に示す場合です。

- OpenTP1 が正常終了したとき
- OpenTP1 の稼働中に，SPP のユーザサーバ名を指定した dcsvstop コマンドを実行したとき

上記のどちらかの事象が起こると，メイン関数で呼び出した dc_rpc_mainloop 関数がリターンして，SPP は終了します。

SPP のプロセスを，kill コマンドで終了させないでください。

(5) SPP の処理の概要

SPP のメイン関数では，次に示す関数を呼び出してください。

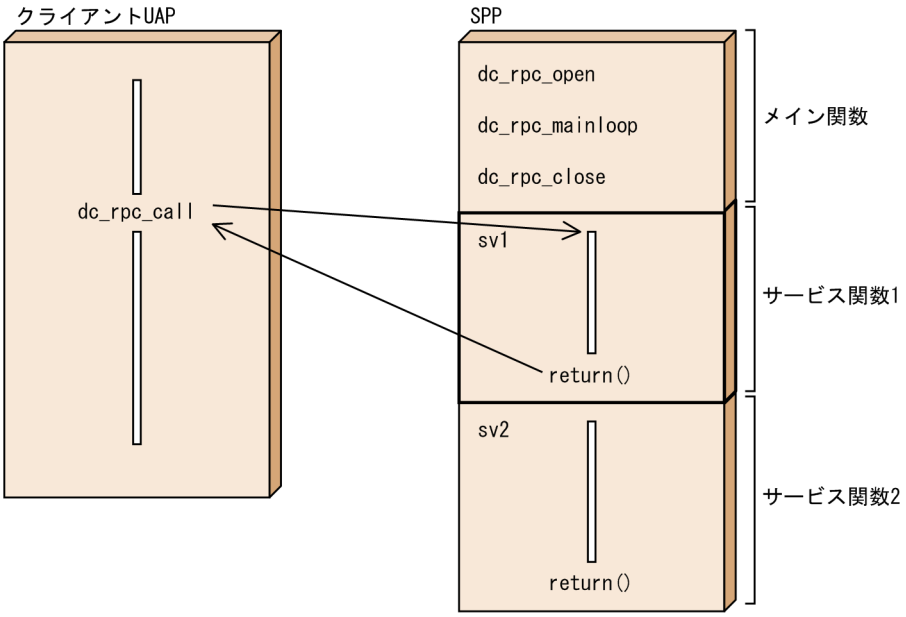
- アプリケーションプログラムの開始（dc_rpc_open 関数【CBLDCRPC('OPEN ')】）
- SPP のサービス開始（dc_rpc_mainloop 関数【CBLDCRSV('MAINLOOP')】）

SPP でトランザクションを開始しているときは，トランザクションをコミット（同期点を取得）してから，SPP を終了させてください。

また，SPP から MCF の関数を呼び出すときは，メイン関数で MCF 環境のオープン（dc_mcf_open 関数【CBLDCMCF('OPEN ')】）と MCF 環境のクローズ（dc_mcf_close 関数【CBLDCMCF('CLOSE ')】）を呼び出してください。

SPP の処理の概要を次の図に示します。

図 1-12　SPP の処理の概要（C 言語の例）



1.2.3　メッセージを処理する UAP（MHP）

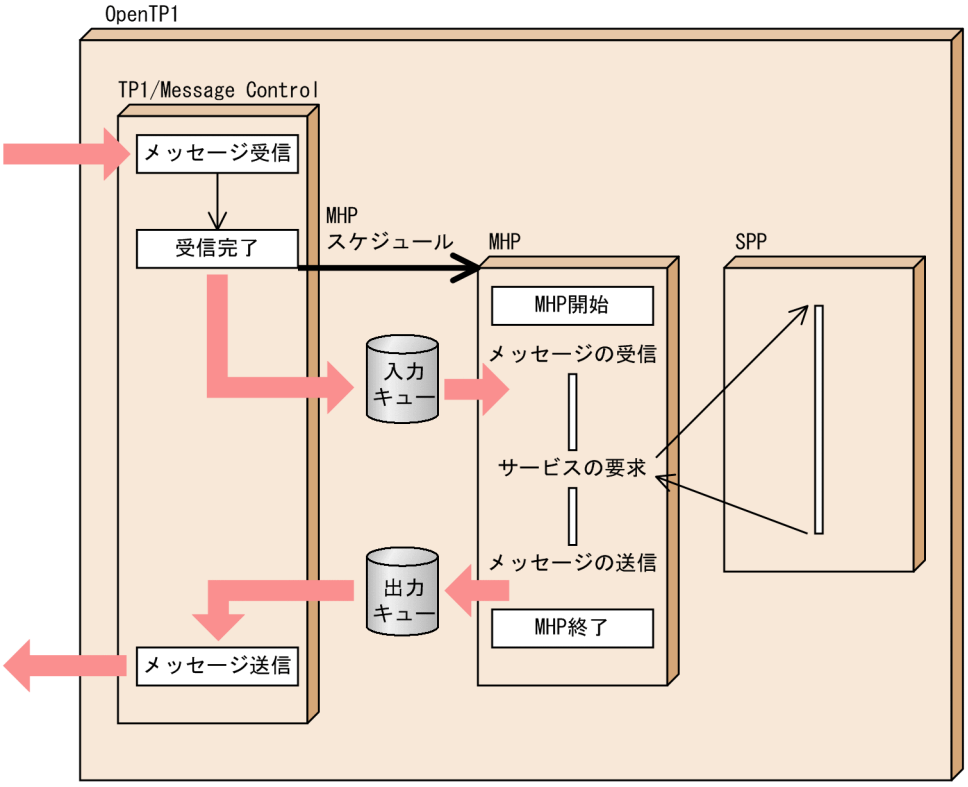
メッセージ送受信で使う UAP を**メッセージ処理プログラム**（MHP）といいます。MHP を使うと，MCF と接続されている他システムとメッセージ送受信形態で通信できます。メッセージ送受信については，「3.6 メッセージ送受信」を参照してください。

MHP では，OpenTP1 のメッセージ送受信機能の関数を使えます。また，MHP の処理から RPC を使って SPP のサービスを要求できます。

MHP は，同じノードに MCF があることが前提です。

MHP の概要を次の図に示します。

図 1-13　MHP の概要



(1)　MHP の構成

MHP は，SPP と同様にメイン関数とサービス関数から構成されます。

MCF で受信したメッセージにある**アプリケーション名**によってスケジュールされる**アプリケーション**を，サービス関数（COBOL 言語の場合はサービスプログラム）として作成します。このサービス関数を複数作成して，メイン関数（COBOL 言語の場合はメインプログラム）で一つの実行形式ファイルにまとめます。そして，一つのメイン関数と複数のサービス関数から構成される MHP の実行形式ファイルを，サービスグループとして OpenTP1 に定義します。

サービス関数動的ローディング機能は，複数のサービスを UAP 共用ライブラリ化※して使うため，複数のサービスをメイン関数にまとめる作業は不要です。

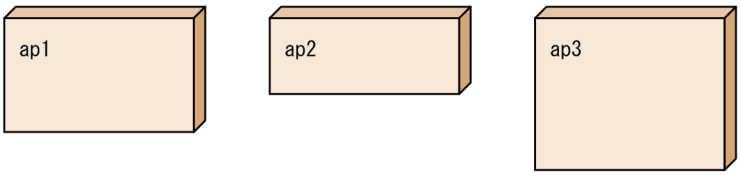
注※

UAP 共用ライブラリ化とは，UAP のソースファイルを翻訳（コンパイル）して作成した UAP オブジェクトファイルを結合（リンケージ）して，共用ライブラリとしてまとめることです。

MHP の構成を，スタブを使う場合とサービス関数動的ローディング機能を使う場合に分けて，それぞれ以降の図に示します。

図 1-14　MHP の構成（スタブを使う場合）

1. クライアントUAPに提供するサービスを，サービス関数として作成します。



2. メイン関数を作成して，メイン関数およびサービス関数を翻訳・結合すると，
MHPの実行形式ファイルとなります。

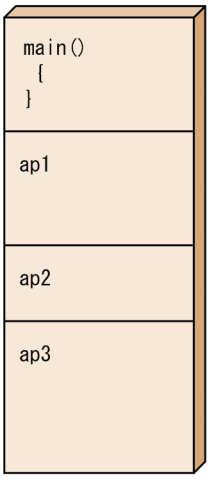
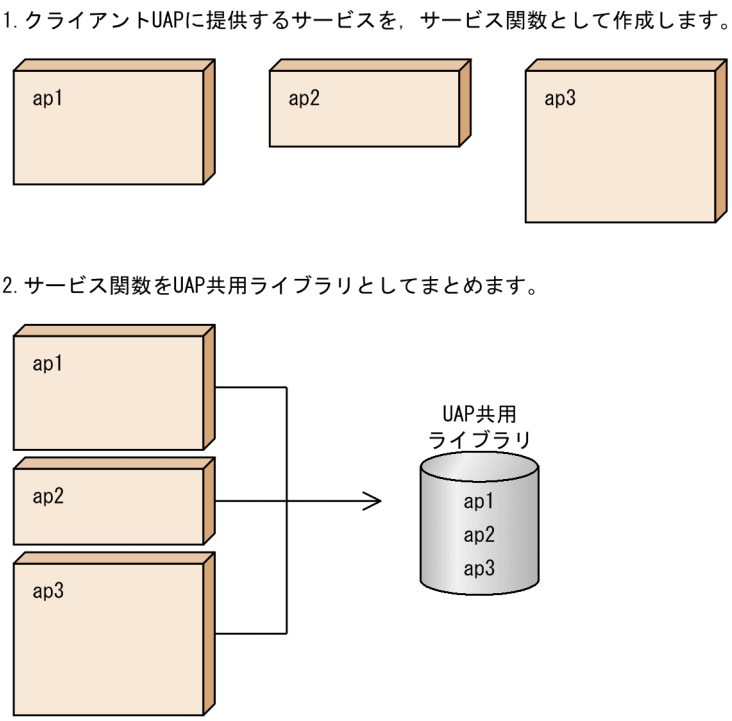


図 1-15　MHP の構成（サービス関数動的ローディング機能を使う場合）



(2)　MHP の開始

MHP を実行する場合，**OpenTP1 の開始と一緒に開始する方法**と，**OpenTP1 の開始後に任意に開始する方法**の 2 とおりがあります。OpenTP1 の開始と一緒に開始すると，OpenTP1 の開始と同時に，MHP の業務を開始できます。MHP の業務内容に応じて，開始する時期を選べます。

(a)　OpenTP1 の開始と一緒に開始する場合

OpenTP1 を開始する前に，OpenTP1 と一緒に開始する指定をしておきます。**ユーザ**サービス構成定義の **dcsvstart 定義コマンド**に，開始する MHP のユーザサーバ名を指定します。

(b)　OpenTP1 の開始後に任意に開始する場合

OpenTP1 の開始後に任意に開始する場合は，**dcsvstart コマンド**の引数に **MHP のユーザサーバ名**を指定して実行します。

MHP はメイン関数から開始します。メイン関数で呼び出す，MHP のサービス開始の関数（dc_mcf_mainloop 関数【CBLDCMCF('MAINLOOP')】）が正常に実行されたことで，メッセージを受信できる状態になります。なお，サービス開始の関数が呼び出される前に MHP が異常終了した場合，該当するユーザサービス定義（またはユーザサービスデフォルト定義）の hold オペランドおよび term_watch_time オペランドの指定値に従って処理します。

(3) MHP の稼働中

開始させた MHP は，メモリを効率的に使うため，事前に指定したプロセスの状態で稼働しています。開始させた MHP を**常駐プロセス**で稼働させる場合と，**非常駐プロセス**で稼働させる場合があります。常駐プロセスとした場合は，サービス要求が来ると MHP の処理を開始します。非常駐プロセスとしてある場合でも，サービス要求が来るとプロセスを自動的に起動して MHP の処理を開始します。

UAP プロセスに関する設定内容については，「1.3.5 アプリケーションプログラムの環境設定」を参照してください。

(a) メッセージを処理する MHP の業務の開始

MCF でメッセージを受信したあとに，該当する MHP のプロセスが起動されます。MHP はメッセージの先頭セグメントにある**アプリケーション名**でスケジュールされます。OpenTP1 内で UAP のサービスを認識するサービス名と，アプリケーション名は，**MCF アプリケーション定義**で対応付けます。

ほかの UAP（MHP または SPP）から dc_mcf_execap 関数【CBLDCMCF('EXECAP ')】で MHP を起動させた場合には，次の 2 とおりの開始方法があります。

- dc_mcf_execap 関数を呼び出した UAP が正常終了（トランザクションがコミット）してから開始。
- UAP が dc_mcf_execap 関数を呼び出した直後から，設定した秒数が過ぎたあと，または設定した時刻になったら開始。

(b) MCF イベント処理用 MHP の業務の開始

MCF の障害や MHP の障害が起こった場合，MCF からエラー内容のデータを通知するためにメッセージが出力されます。これを **MCF イベント**といいます。MCF イベントが通知された場合に備えて，**MCF イベント処理用 MHP** を作成しておくと，独自の障害対策処理ができます。MCF イベント処理用 MHP は，通知される MCF イベントのイベントコードに対応させて作成します。該当する MCF イベントが通知されたときに，MCF イベント処理用 MHP が起動されます。MCF イベントについては，「3.10 MCF イベント」を参照してください。

(4) MHP の終了

MHP が正常終了するのは，次に示す場合です。

- OpenTP1 が正常終了した場合
- OpenTP1 の稼働中に，MHP のユーザサーバ名を指定した dcsvstop コマンドを実行した場合

上記のどちらかの事象が起こると，メイン関数で呼び出した dc_mcf_mainloop 関数がリターンして，MHP は終了します。

MHP のプロセスを，kill コマンドで終了させないでください。

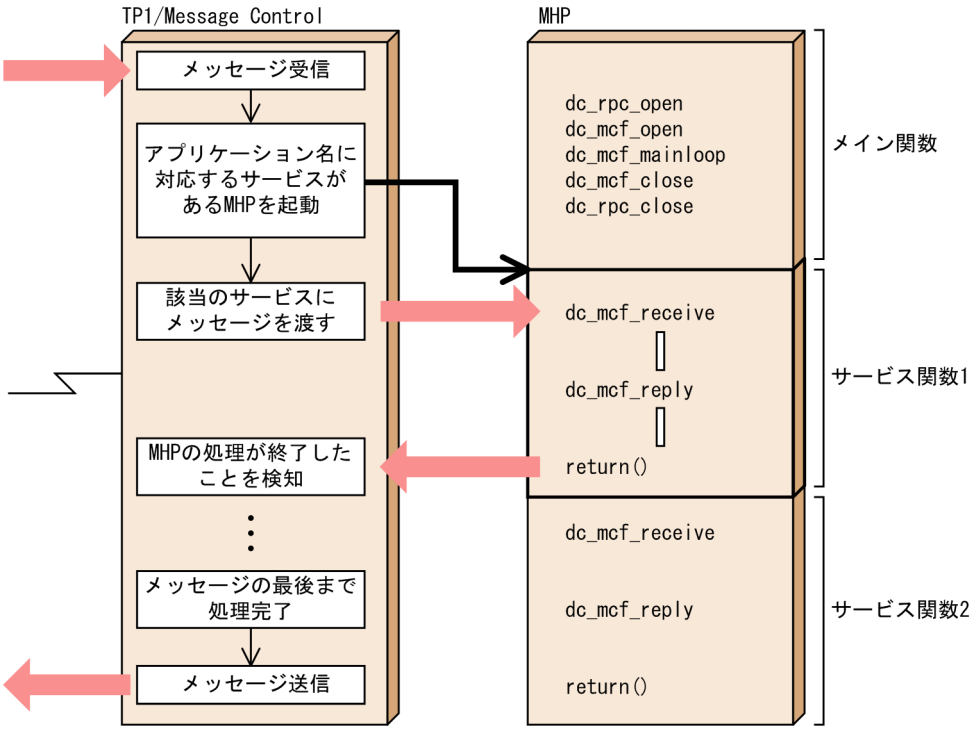
(5) MHP の処理の概要

MHP のメイン関数では，次に示す関数を呼び出してください。

- アプリケーションプログラムの開始（dc_rpc_open 関数【CBLDCRPC('OPEN ')】）
- MCF 環境のオープン（dc_mcf_open 関数【CBLDCMCF('OPEN ')】）
- MHP のサービス開始（dc_mcf_mainloop 関数【CBLDCMCF('MAINLOOP')】）
- MCF 環境のクローズ（dc_mcf_close 関数【CBLDCMCF('CLOSE ')】）

MHP の処理の概要を次の図に示します。

図 1-16 MHP の処理の概要（C 言語の例）



(6) 注意事項

MHP のサービス関数を RPC で呼び出すことはできません。

1.2.4　オフラインの業務をするUAP

オフラインのバッチ業務をするUAPを任意に作成できます。バッチ業務のほかにDAMファイルの初期化，DAMファイルの割り当てや削除の処理をするUAPは，オフライン環境で実行します。

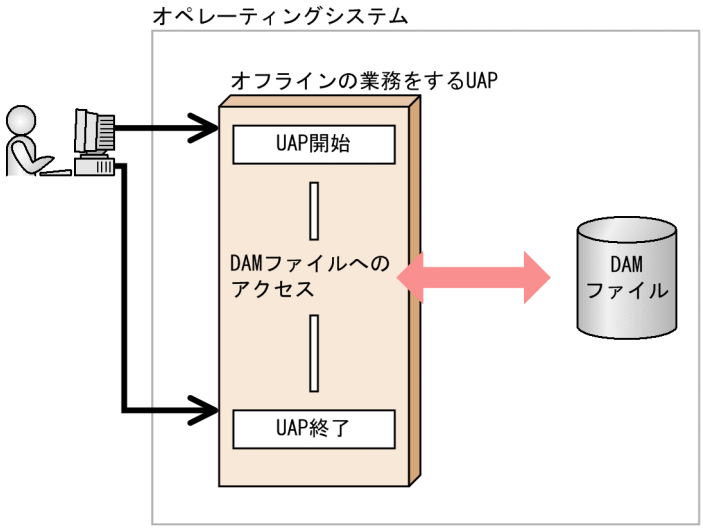
オフラインの業務をするUAPで使えるOpenTP1の機能を次に示します。

- DAMファイルの物理ファイルに対する処理
- jnlrputコマンド出力ファイルのジャーナルデータの編集

オフラインの業務をするUAPでは，オンラインで使うOpenTP1の関数は使えません。また，オフラインの業務をするUAPとオンライン環境のUAP（SUP，SPP，MHP）では，サービス要求などの通信はできません。オフラインの業務をするUAPに，ほかのUAPに提供するサービスは作成できません。

オフラインの業務をするUAPの概要を次の図に示します。

図 1-17　オフラインの業務をするUAPの概要



(1)　オフラインの業務をするUAPの開始／終了

オフラインの業務をするUAPは，シェルから開始します。オフラインの業務をするUAPの開始と終了はユーザで管理します。

1.3　アプリケーションプログラムの作成

OpenTP1 の UAP を作成する手順について説明します。UAP の作成手順を次の図に示します。

図 1-18　UAP の作成手順（スタブを使う場合）

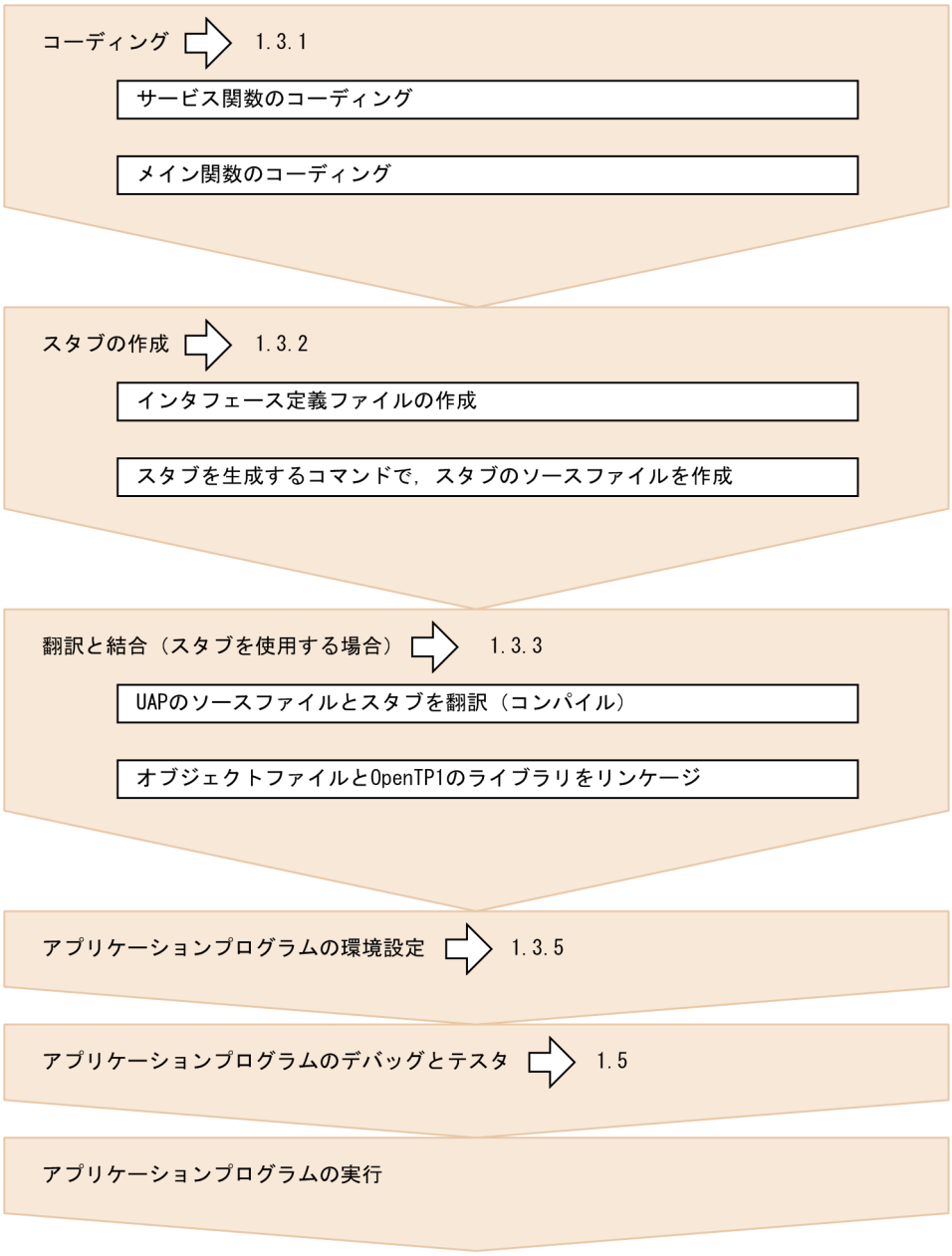
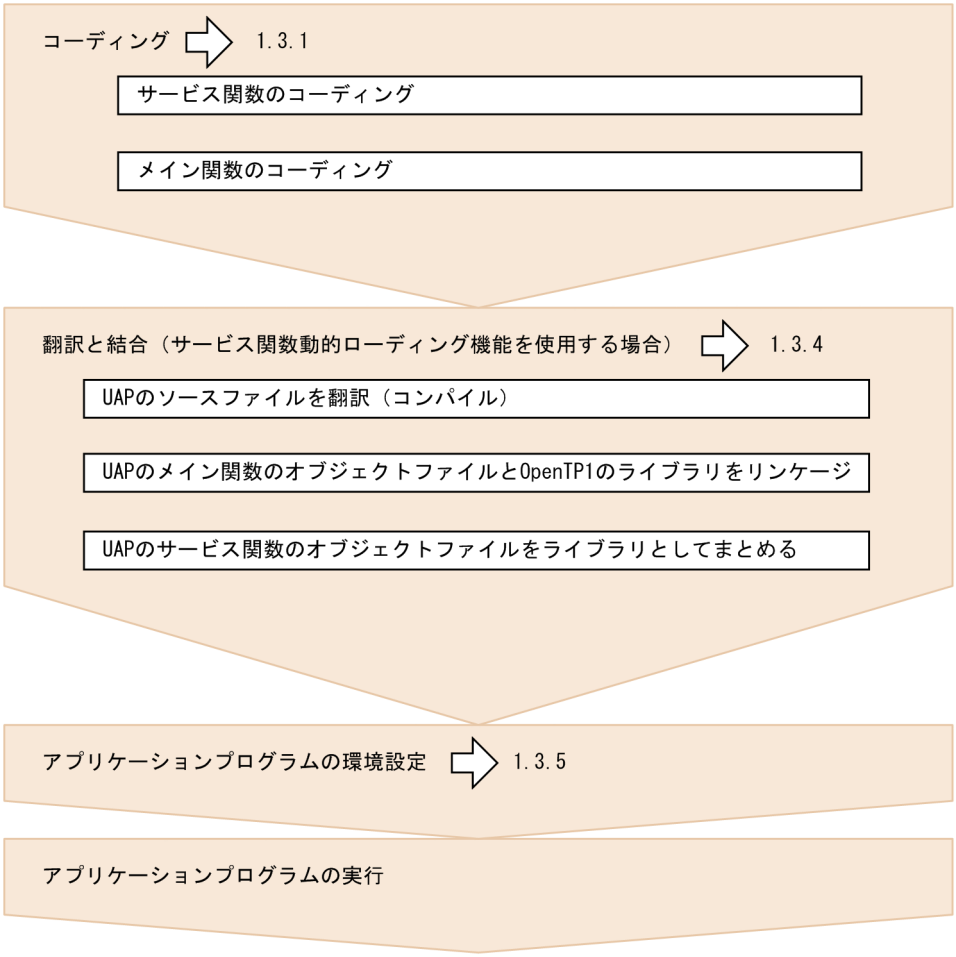


図 1-19　UAP の作成手順（サービス関数動的ローディング機能を使う場合）



1.3.1　コーディング

OpenTP1 の UAP をコーディングする場合，C 言語，C++言語，または COBOL 言語を使います。UAP では，OpenTP1 の機能のほかにも，OS の標準の機能やデータベース言語（SQL）を使えます。コーディングの規約については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。SQL のコーディングの規約については，該当するリファレンスマニュアルを参照してください。

(1)　C 言語または C++言語でコーディングする場合

(a)　C 言語を使うとき

ANSI C の形式，ANSI 準拠前の K&R の形式（Classic C）のどちらかに従ってコーディングします。UAP で OpenTP1 の機能を使うときは，OpenTP1 のライブラリ関数を呼び出します。

(b) C++言語を使うとき

ANSI C の形式で C++言語の仕様に従ってコーディングします。UAP で OpenTP1 の機能を使うときは，OpenTP1 のライブラリ関数を呼び出します。OpenTP1 のライブラリ関数は，ヘッダファイル（dc×××.h）で C 言語のリンケージを指定しているため，C++言語でコーディングした UAP のリンケージの際には C 言語の関数としてリンケージされ動作します。

(c) OpenTP1 の関数の使い方

OS で標準的に提供する関数と同様，関数を呼び出すときには，引数を設定します。

関数が正常に実行されたかどうかは，戻ってくる値（リターン値）でわかります。関数には，リターン値を返す関数と返さない関数があります。

C 言語でコーディングする場合の概要を次の図に示します。

図 1-20 C 言語でコーディングする場合の概要

```
#include <stdio.h>
#include <string.h>
#include <dcrpc.h>
#include <dctrn.h>

main()
{
/* UAPの開始 */

    rc = dc_rpc_open(DCNOFLAGS);
    if(rc != DC_OK){
        printf("dc_rpc_openが失敗 !!");
        goto PROG_END;
    }

/* リモートプロシジャコール  */

    rc = dc_rpc_call("svr01","svr01",…);
    if(rc != DC_OK){
        printf("サービスの要求失敗 !!")
        goto PROG_END;
    }

/* UAPの終了 */
    PROG_END:
    dc_rpc_close(DCNOFLAGS);
    printf("処理を終了しました。¥n");
    exit(0);
}
```

プログラムの処理で使うヘッダファイルを取り込みます。OpenTP1のヘッダファイルは"dc" で始まります。

メイン関数はmain()，サービス関数は関数名で始めます。

リターン値を返すライブラリ関数は，分岐条件を付けることができます。

リターン値がないライブラリ関数は，そのまま呼び出します。

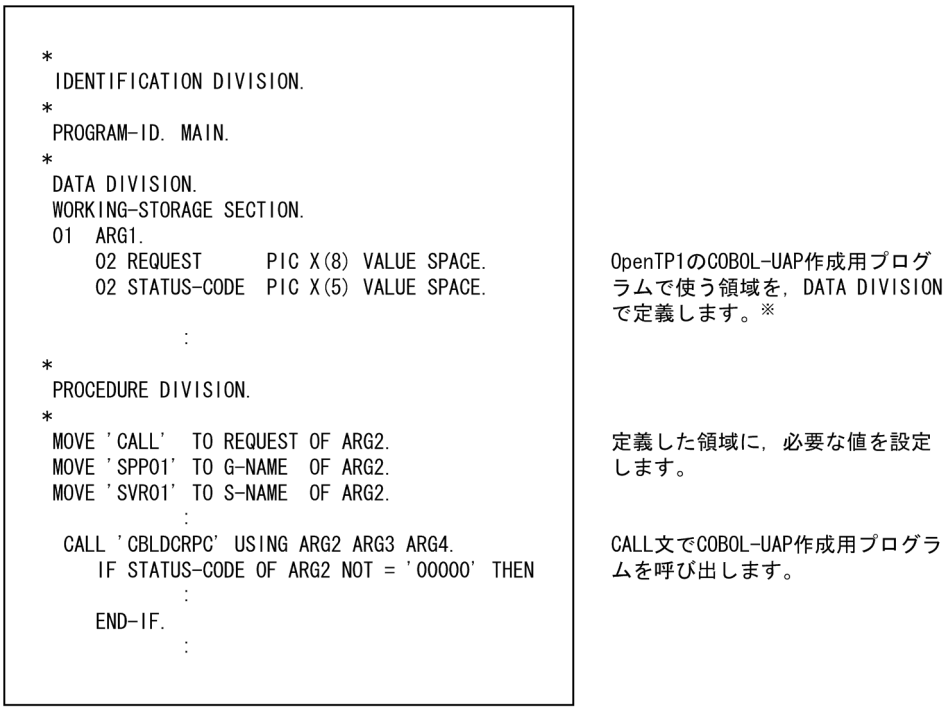
(2) COBOL 言語でコーディングする場合

COBOL 言語を使うときは，COBOL85 または COBOL2002 の形式でコーディングします。UAP から OpenTP1 の機能を使うときは，OpenTP1 のライブラリ関数に対応した COBOL-UAP 作成用プログラムを使います。COBOL-UAP 作成用プログラムを，COBOL の CALL 文で呼び出して，UAP の処理から OpenTP1 のライブラリに制御を移します。

CALL 文の実行結果は，戻ってくる数値（**ステータスコード**）でわかります。COBOL-UAP 作成用プログラムには，ステータスコードを返さないものもあります。

COBOL 言語でコーディングする場合の概要を次の図に示します。

図 1-21　COBOL 言語でコーディングする場合の概要



注※

TP1/Server Base サンプルでは，COBOL-UAP 作成用プログラムごとに，DATA DIVISION の**テンプレート**（ひな形）を使えます。DATA DIVISION のテンプレートについては，「8.8　COBOL 言語用テンプレート」を参照してください。

(3)　注意事項

- JNI（Java Native Interface）は使用できません。使用した場合の動作は保証できません。
- OpenTP1 の API は，すべてメインスレッド上で発行してください。UAP はマルチスレッドで作成できますが，OpenTP1 が提供する API はスレッドセーフではないためです。メインスレッド以外で OpenTP1 の API を発行すると，UAP が誤動作して異常終了するおそれがあります。ただし，次に示す API は，メインスレッド以外でも発行できます。
  - ee_で始まる TP1/EE の C 言語インタフェース
  - CBLEE で始まる TP1/EE の COBOL 言語インタフェース

1.3.2　スタブの作成

OpenTP1 で使う UAP を作成するときには，UAP 間でサービスを要求できるようにするライブラリが必要となります。このライブラリを**スタブ**といいます。スタブには，UAP のサービスに関する情報を指定します。また，通信相手の情報を作成する場合もあります。

スタブの作成方法については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。

(1)　アプリケーションプログラムに結合させるスタブの種類

スタブには，サーバ UAP に結合させるスタブとクライアント UAP に結合させるスタブがあります。

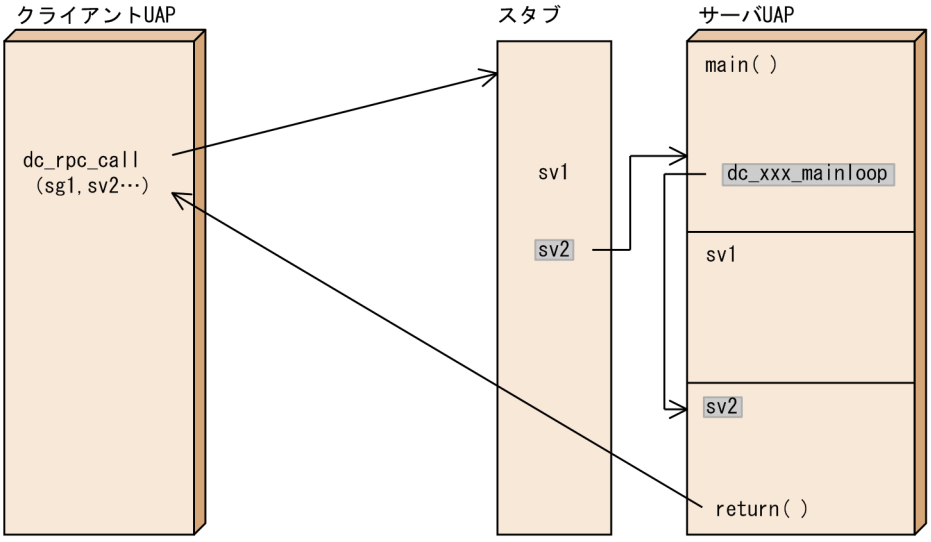
(a)　サーバ UAP に結合させるスタブ

サーバ UAP に結合させるスタブは，サービスを振り分ける関数と連動して UAP のサービスを実行できるようにします。サービスを振り分ける関数は，サーバ UAP のメイン関数で呼び出します。サービスを振り分ける関数を，次に示します。

- SPP の場合：dc_rpc_mainloop 関数【CBLDCRSV('MAINLOOP')】
- MHP の場合：dc_mcf_mainloop 関数【CBLDCMCF('MAINLOOP')】

サーバ UAP に結合させるスタブの概要を次の図に示します。

図 1-22　サーバ UAP に結合させるスタブの概要



(b)　クライアント UAP に結合させるスタブ

クライアント UAP に結合させるスタブは，サーバ UAP の情報を指定することで，サーバ UAP と通信できるようにします。クライアント UAP にスタブが必要になるのは，XATMI インタフェースを使った通信の場合だけです。OpenTP1 の RPC を使う場合は，クライアント UAP にスタブは必要ありません。

(2) スタブが必要な UAP

UAP にスタブが必要かどうかは，UAP の種類や通信方法によって異なります。

- **OpenTP1 のリモートプロシジャコールを使う UAP（SUP，SPP）**
  SPP にスタブが必要です。SUP にはスタブは必要ありません。
- **MHP の場合**
  MHP にはスタブが必要です。SPP と同様の手順でスタブを作成します。
- **XATMI インタフェースを使ってクライアント／サーバ形態の通信をする場合**
  クライアント UAP（SUP，SPP）とサーバ UAP（SPP）の両方に，スタブが必要です。

オフラインの業務をする UAP は，サービス関数がないので，スタブを作成する必要はありません。

(3) スタブの作成手順

スタブを作成するときは，UAP に関する情報の定義を格納したファイル（**RPC インタフェース定義ファイル**※）を作成します。そして，RPC インタフェース定義ファイルを引数にして，**スタブを生成するコマンド**を実行します。スタブを生成するコマンドを，次に示します。

- TCP/IP 通信をする UAP の場合：**stbmake コマンド**
- OSI TP 通信をする UAP の場合：**tpstbmk コマンド**

スタブを生成するコマンドを実行すると，**スタブのソースファイル**（C 言語のソースファイル）が作成されます。このファイルを C 言語のコンパイラで翻訳して，UAP のオブジェクトファイルに結合させます。

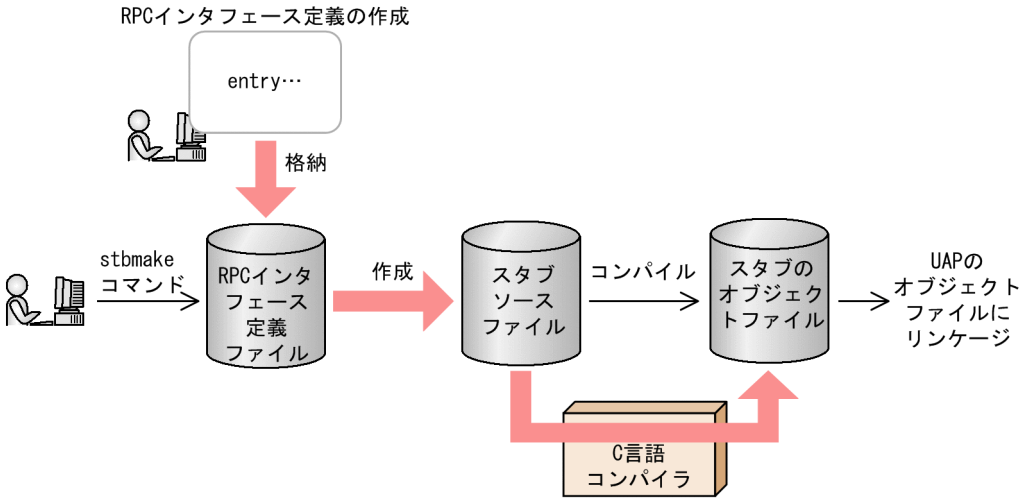
MHP を ANSI C 形式，および C++形式でコーディングした場合，その MHP に結合するスタブの翻訳時には，**DCMHP** を定義してください。

スタブの内容を変更するときは，UAP を作成する一連の作業をやり直します。定義ファイルの内容を変更して，スタブを作り直してから，コンパイルし直した UAP のオブジェクトファイルに結合させてください。

注※
XATMI インタフェースのスタブの場合は，**XATMI インタフェース定義ファイル**といいます。

スタブの作成手順を次の図に示します。

図 1-23　スタブの作成手順



1.3.3　翻訳と結合（スタブを使用する場合）

作成したプログラムの翻訳（コンパイル）と結合（リンケージ）の手順について説明します。UAP をコンパイル，リンケージして，実行形式ファイルとします。コンパイルとリンケージの手順については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。

(1)　翻訳（コンパイル）

コンパイルするプログラムを次に示します。

- **UAP のソースファイル**（メイン関数，サービス関数）
- **スタブ**（UAP に必要な場合）

C 言語のプログラムは C 言語のコンパイラで，COBOL 言語で作成したプログラムは該当する COBOL 言語のコンパイラで翻訳します。

(2)　結合（リンケージ）

コンパイルして作成したオブジェクトファイルを，OpenTP1 のライブラリなど必要なファイルとリンケージします。OpenTP1 以外の RM を使う場合は，OpenTP1 以外の RM が指定するライブラリとリンケージします。OpenTP1 以外の RM を XA インタフェースで使う場合，次の手順で UAP にライブラリをリンケージします。

1. OpenTP1 以外の RM のリソースマネジャ識別子を trnmkobj コマンドに指定して，トランザクション制御用オブジェクトファイルを作成します。
2. 作成したトランザクション制御用オブジェクトファイルを，UAP にリンケージします。

(3) 注意事項

OS が HP-UX の場合，リンケージ時のバインドモードには必ず"immediate"を指定してください。"immediate"以外のバインドモードで作成した実行形式ファイルを OpenTP1 の UAP として使った場合，システムの動作は保証しません。作成した UAP のバインドモードが"immediate"かどうかは，OS の chatr コマンドで確認してください。

1.3.4 翻訳と結合（サービス関数動的ローディング機能を使用する場合）

作成したプログラムの翻訳（コンパイル），結合（リンケージ），および UAP 共用ライブラリ化※の手順について説明します。

まず，UAP のメイン関数をコンパイル，リンケージして実行形式ファイルとします。次に，UAP のサービス関数を UAP 共用ライブラリ化※します。コンパイルとリンケージの手順については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。

注※

UAP 共用ライブラリ化とは，UAP のソースファイルを翻訳（コンパイル）して作成した UAP オブジェクトファイルを結合（リンケージ）して，共用ライブラリとしてまとめることです。

(1) 翻訳（コンパイル）

コンパイルするプログラムを次に示します。

- UAP のソースファイル（メイン関数，サービス関数）

C 言語のプログラムは C 言語のコンパイラで，COBOL 言語で作成したプログラムは該当する COBOL 言語のコンパイラで翻訳します。

(2) 結合（リンケージ）

UAP のメイン関数のソースファイルをコンパイルして作成したオブジェクトファイルを，OpenTP1 のライブラリなど必要なファイルとリンケージします。OpenTP1 以外の RM を使う場合は，OpenTP1 以外の RM が指定するライブラリとリンケージします。OpenTP1 以外の RM を XA インタフェースで使う場合，次の手順で UAP にライブラリをリンケージします。

1. OpenTP1 以外の RM のリソースマネジャ識別子を trnmkobj コマンドに指定して，トランザクション制御用オブジェクトファイルを作成します。
2. 作成したトランザクション制御用オブジェクトファイルを，UAP にリンケージします。

(3) UAP 共用ライブラリの作成

UAP のサービス関数のソースファイルをコンパイルして作成したオブジェクトファイルを，共用ライブラリ化します。このとき，(2)と同様に，OpenTP1 のライブラリなど必要なファイルとリンケージします。コンパイルオプションおよびリンケージオプションについては，TP1/Server Base サンプル（make_svdl ファイル）を参照してください。

(4) 注意事項

OS が HP-UX の場合，リンケージ時のバインドモードには必ず"immediate"を指定してください。"immediate"以外のバインドモードで作成した実行形式ファイルを OpenTP1 の UAP として使った場合，システムの動作は保証しません。作成した UAP のバインドモードが"immediate"かどうかは，OS の chatr コマンドで確認してください。

1.3.5 アプリケーションプログラムの環境設定

作成した UAP の実行形式ファイルを，OpenTP1 のユーザサーバとして使えるように，環境設定します。

(1) UAP を格納するディレクトリ

作成した UAP の実行形式ファイルは，**$DCDIR/aplib/**ディレクトリ（$DCDIR は，OpenTP1 ホームディレクトリを示します）に格納します。

(2) UAP の登録

UAP の実行形式ファイルを，OpenTP1 に登録します。OpenTP1 の UAP は，サービスを提供することから**ユーザサーバ**といいます。

(a) UAP の登録方法

OpenTP1 に UAP を登録するときに，実行環境を設定します。実行環境を設定する方法を次に示します。

- TP1/Server Base の場合
  ユーザサービス定義で設定します。
- TP1/LiNK の場合
  UNIX の場合は **dcsysset -u コマンド**を，Windows の場合は**［アプリケーション管理］アイコン**を使います。

(b) ユーザサーバ名

OpenTP1 で UAP を操作するときに使う名称を**ユーザサーバ名**といいます。ユーザサーバ名は，次のようになります。

- TP1/Server Base の場合

ユーザサービス定義のファイル名です。

- **TP1/LiNK の場合**

  UAP の環境を設定するときに，**実行形式ファイル名**に対応させてユーザサーバ名を任意で付けます。

(3) UAP の名称の関係

UAP として作成するプログラムの名称について説明します。

- **UAP の実行形式ファイル名**

  UAP のオブジェクトファイルをライブラリとリンケージするときに，リンケージのコマンドにオプションで設定した名称です。

- **ユーザサーバ名**

  UAP を OpenTP1 に登録するときに設定した名称です。dcsvstart コマンドの引数に使用します。ユーザサーバ名は，1～8 文字で設定します。

- **サービスグループ名，サービス名**

  OpenTP1 のリモートプロシジャコールや XATMI インタフェースの通信で，サービスを要求する関数の引数に設定する名称です。ユーザサーバ名を OpenTP1 に登録するときに一緒に指定します。

  サービスグループ名は，UAP の実行形式ファイル単位に名称を付けます。

  サービス名は，サービス関数の関数名です。

- **アプリケーション名**

  TP1/Message Control で受け取ったメッセージを処理するときに，該当するメッセージを処理するアプリケーションであることを識別する名称です。MHP のサービス関数をアプリケーション名として登録します。サービス名とアプリケーション名の対応は MCF アプリケーション定義に指定します。

1.3.6 ユーザサーバの負荷分散とスケジュール

UAP（ユーザサーバ）を効率良く使うための機能（マルチサーバ）と UAP のスケジュール方法について説明します。

OpenTP1 のシステムサービス，またはユーザサーバを実行するときには，OS の作業領域を使用します。この作業領域の処理を**プロセス**といいます。ユーザサーバを実行して生成されるプロセスを特に**ユーザサーバプロセス**，**UAP プロセス**，または単に**プロセス**といいます。OpenTP1 では，プロセスが必要以上に増えたり減ったりしないように，使うプロセスの総数を制御しています。

ユーザサーバのプロセスを制御するには，**ユーザサーバを事前に開始していることが前提**です。OpenTP1 と一緒に開始させておくか，dcsvstart -u コマンドで開始させておきます。

(1) マルチサーバ

実行中のユーザサーバに対して，さらにサービス要求が来た場合でも，ユーザサーバの処理を新しい別のプロセスで実行できます。このように，一つのユーザサーバの処理を，別のプロセスで並行して実行する機能を**マルチサーバ**といいます。

マルチサーバを使えるのは，スケジュールキューを使う SPP（**キュー受信型サーバ**）です。**ソケット受信型サーバ**の場合はマルチサーバを使えません。ソケット受信型サーバには，使うプロセスは一つだけと指定してください。

(2) 常駐プロセスと非常駐プロセス

マルチサーバを指定した UAP のプロセスを，OpenTP1 の稼働中にいつも確保しておくことも，動的に確保することもできます。いつも確保されているプロセスを**常駐プロセス**といいます。また，稼働中には確保されていなくて，必要に応じて起動されるプロセスを**非常駐プロセス**といいます。

プロセスを非常駐プロセスと設定しておくと，OpenTP1 システム内のメモリ領域を効率良く使えます。また，プロセスを常駐プロセスと設定しておくと，そのユーザサーバの処理は非常駐プロセスに比べて速くなります。

システムのメモリに空きがない場合，非常駐プロセスは稼働中の非常駐プロセスが終了してから実行されます。

非常駐プロセスを使用すると，一時的※に最大でユーザサービス定義の parallel_count オペランドの指定値の 2 倍のプロセス数が起動されることがあります。

注※

終了しようとしているプロセスが，dc_rpc_mainloop 関数または dc_mcf_mainloop 関数の終了後から dc_rpc_close 関数終了までの区間にある場合で，新たなサービス要求を受け付けたタイミングです。

また，非常駐プロセスであってもプロセス起動による性能への影響を最小限にとどめるため，同一プロセスで続けてサービス要求を処理する場合があります。そのため，ユーザサーバはリエントラントできるプログラム構造で作成する必要があります。

(3) プロセスの設定方法

プロセスを常駐とするか非常駐とするかは，ユーザサーバの起動プロセス数で，事前に設定しておきます。設定したプロセスの数だけ，並行にプロセスを起動できます。設定する方法を次に示します。

- TP1/Server Base の場合

  ユーザサービス定義の parallel_count オペランドで，使うプロセスの合計（常駐プロセス数と非常駐プロセス数）を設定します。
- TP1/LiNK の場合

UAP の実行環境を設定するときに，使うプロセスの数（常駐プロセス数と非常駐プロセス数）を設定します。

常駐プロセスを複数個指定していれば，指定した数だけ並行してプロセスが起動されます。また，非常駐プロセスを複数個指定していれば，指定した数だけ動的にプロセスが起動されます。

(4) マルチサーバ負荷バランス

スケジュールキューにあるサービスの要求数に応じて，非常駐プロセスの数を増やしたり減らしたりする機能を**マルチサーバ負荷バランス機能**といいます。

非常駐プロセスをいつ起動するかは，ユーザサービス定義の balance_count オペランドに指定する値で決まります。（balance_count オペランドに指定した値×起動中のプロセス）の数を超えてサービス要求が滞留したときに，OpenTP1 は非常駐プロセスを起動します。スケジュールキューに滞留しているサービス要求の数が 0 になると，OpenTP1 は非常駐プロセスを終了させます。

非常駐プロセスを起動するタイミングの値を指定する方法を次に示します。

- TP1/Server Base の場合

  ユーザサービス定義の balance_count オペランドに設定します。
- TP1/LiNK の場合

  UAP の実行環境を設定するときのサービスの最大滞留数が，上記の balance_count オペランドの値になります。

(5) 非常駐 UAP プロセスのリフレッシュ機能

非常駐プロセスを使用する場合，一つのサービス要求ごとに実行プロセスを起動し直すことができます。この機能を，**非常駐 UAP プロセスのリフレッシュ機能**といいます。この機能を使用することで，リエントラント構造ではない UAP の実行もできるようになります。

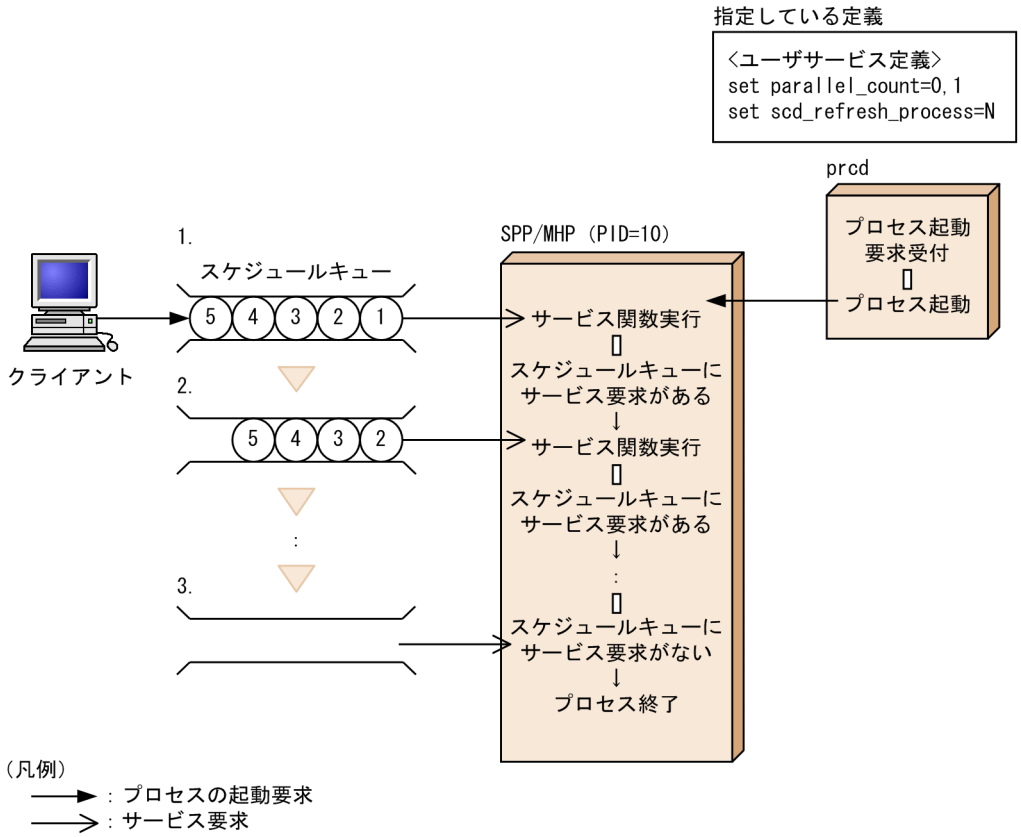
この機能を使用するためには，ユーザサービス定義またはユーザサービスデフォルト定義に，scd_refresh_process オペランドを指定します。また，この機能は 1 プロセスが処理するサービス要求数（ユーザサービス定義の balance_count オペランドの指定値）が 0 の場合にだけ使用できます。

非常駐 UAP プロセスのリフレッシュ機能を使用しない場合と使用する場合の動作，およびこの機能を使用する場合の注意事項について説明します。

(a) 非常駐 UAP プロセスのリフレッシュ機能を使用しない場合の動作

非常駐 UAP プロセスのリフレッシュ機能を使用しない場合の動作を，次の図に示します。図中の番号と以降の説明の番号は対応しています。

図 1-24　非常駐 UAP プロセスのリフレッシュ機能を使用しない場合の動作



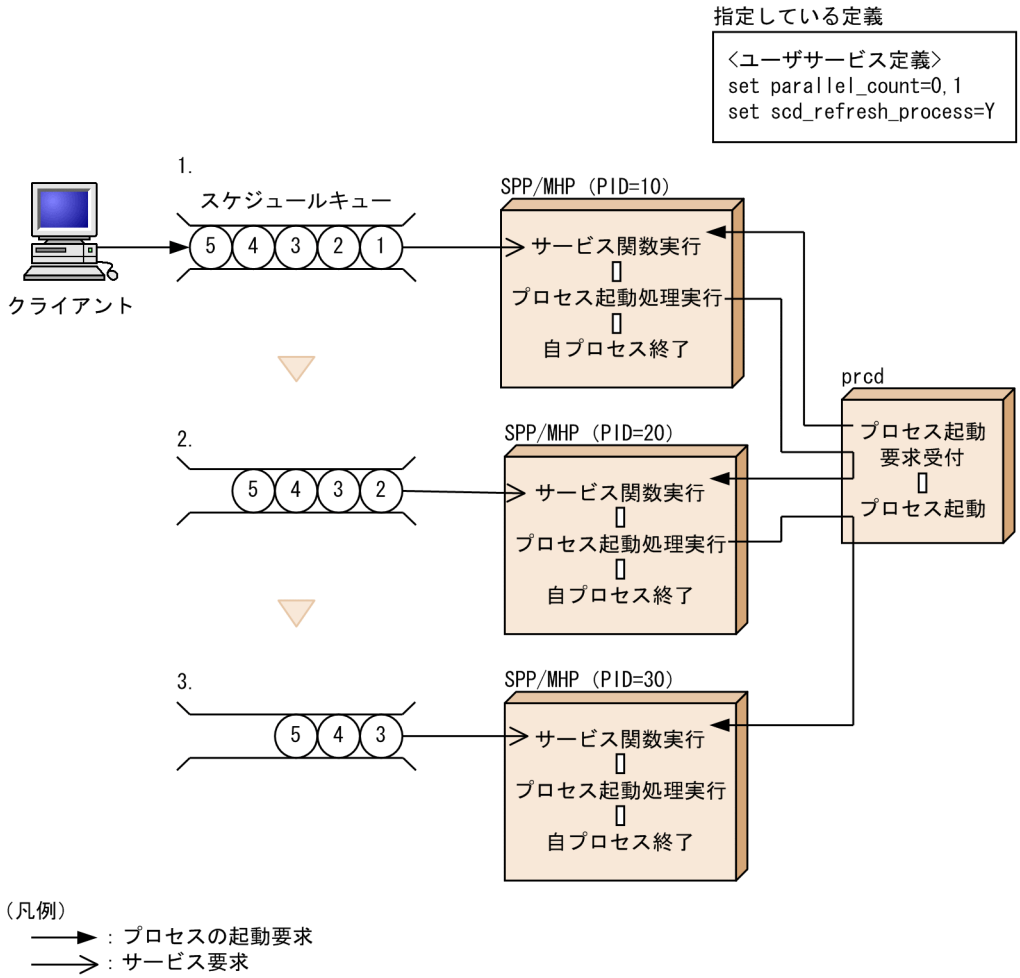
1. SPP または MHP のスケジュールキューにサービス要求が登録されたため，プロセス ID が 10 の SPP または MHP プロセスを起動します。また，一つ目のサービス要求について，サービス関数を実行します。
2. サービス関数の実行後，SPP または MHP のスケジュールキューにサービス要求がある場合は，そのプロセス上で再度サービス関数を実行します。

   以降，スケジュールキューからサービス要求がなくなるまで，処理を繰り返します。
3. SPP または MHP のスケジュールキューにサービス要求がなくなったので，プロセスが終了となります。

(b)　非常駐 UAP プロセスのリフレッシュ機能を使用する場合の動作

非常駐 UAP プロセスのリフレッシュ機能を使用する場合の動作を，次の図に示します。図中の番号と以降の説明の番号は対応しています。

図 1-25　非常駐 UAP プロセスのリフレッシュ機能を使用する場合の動作



1. SPP または MHP のスケジュールキューにサービス要求が登録されたため，プロセス ID が 10 の SPP または MHP プロセスを起動します。また，一つ目のサービス要求についてサービス関数を実行したあと，プロセス起動処理を実行して自プロセスは終了します。
2. プロセス ID が 10 の SPP または MHP で実行したプロセス起動処理によって，prcd はプロセス ID が 20 の SPP または MHP プロセスを起動します。また，二つ目のサービス要求についてサービス関数を実行したあと，プロセス起動処理を実行して自プロセスは終了します。
3. プロセス ID が 20 の SPP または MHP で実行したプロセス起動処理によって，prcd はプロセス ID が 30 の SPP または MHP プロセスを起動します。また，三つ目のサービス要求についてサービス関数を実行したあと，プロセス起動処理を実行して自プロセスは終了します。

   以降，スケジュールキューからサービス要求がなくなるまで，処理を繰り返します。

(c)　非常駐 UAP プロセスのリフレッシュ機能使用時の注意事項

- 起動した時点で常駐プロセスがある構成（常駐プロセス数に 1 以上を指定する）のサーバを，scdchprc コマンドを使用して非常駐プロセスだけで構成される（常駐プロセス数に 0 を，最大プロセス数に 1 以上を指定する）サーバに変更しても，この機能の対象にはなりません。

- 起動した時点で非常駐プロセスだけで構成される（常駐プロセス数に 0 を，最大プロセス数に 1 以上を指定する）サーバを，scdchprc コマンドを使用して一つ以上の常駐プロセスがある構成（常駐プロセス数に 1 以上を指定する）のサーバに変更した場合は，常駐プロセスおよび非常駐プロセスの両方がこの機能の対象になります。
- OpenTP1 システムで同時に起動されるプロセス数が一時的に増加する可能性があるため，この機能を使用する場合は十分な検証を実施し，必要となる最大同時起動サーバプロセス数（プロセスサービス定義の prc_process_count オペランドの指定値）を見積もってください。

  検証，見積もり方法を次に示します。

  1. 現状の OpenTP1 を起動して次のコマンドを実行し，表示されたサーバの中から「_」付きのサーバ数を数えます。

     ```
     prcls -a
     ```

  2. クライアントサービス定義の parallel_count オペランドの指定値である，最大プロセス数の合計値を 2 倍にした値を算出します。
  3. 手順 1.および 2.の合計値以上の値を，prc_process_count オペランドに設定します。
- システム構成によっては，UAP プロセスの起動と停止が頻繁に発生する可能性があります。その場合はシステム資源（CPU，メモリ，動的ポートなど）が枯渇するおそれがあるため，十分な検証を実施し，システム資源を見積もってください。

  それぞれのシステム資源の検証方法を次に示します。

  - CPU メモリ

    高負荷テストや長時間連動試験中に，パフォーマンスモニタなどで CPU やメモリの状態を定期的に確認し，不足することがないか検証してください。

  - TCP/IP の動的ポート

    OpenTP1 の UAP は，起動時に TCP/IP の動的ポートを使用します。UAP 停止時には動的ポートを開放しますが，TCP/IP の仕様で一定時間（TIME_WAIT 状態の間）資源を確保します。そのため，OS で使用できる動的ポート数を，一定時間内で使い切らないようにしてください。高負荷テストや長時間連動試験中に netstat コマンドを定期的に実行し，OS で使用できる動的ポート数の使用状況を監視して不足していないか検証してください。
- プロセスの起動とサーバマシンのログオフ処理が重なると，OpenTP1 が KFCA01820-E メッセージを出力し，プロセス初期化エラーになることがあります。そのため，UAP プロセスの起動と停止が頻繁に発生するシステムでは，該当するサーバマシンでのログオフ操作を控えてください。
- プログラム言語として COBOL2002 を使用する場合は，コンパイルオプション「-CBLVALUE」を指定したコンパイルで UAP を作成してください。「-CBLVALUE」を指定することでプロセス再起動ごとに VALUE 句の指定がないデータ項目が初期化された状態となります。コンパイルオプションの詳細については，COBOL2002 のマニュアルを参照してください。

(6) スケジュールの優先度

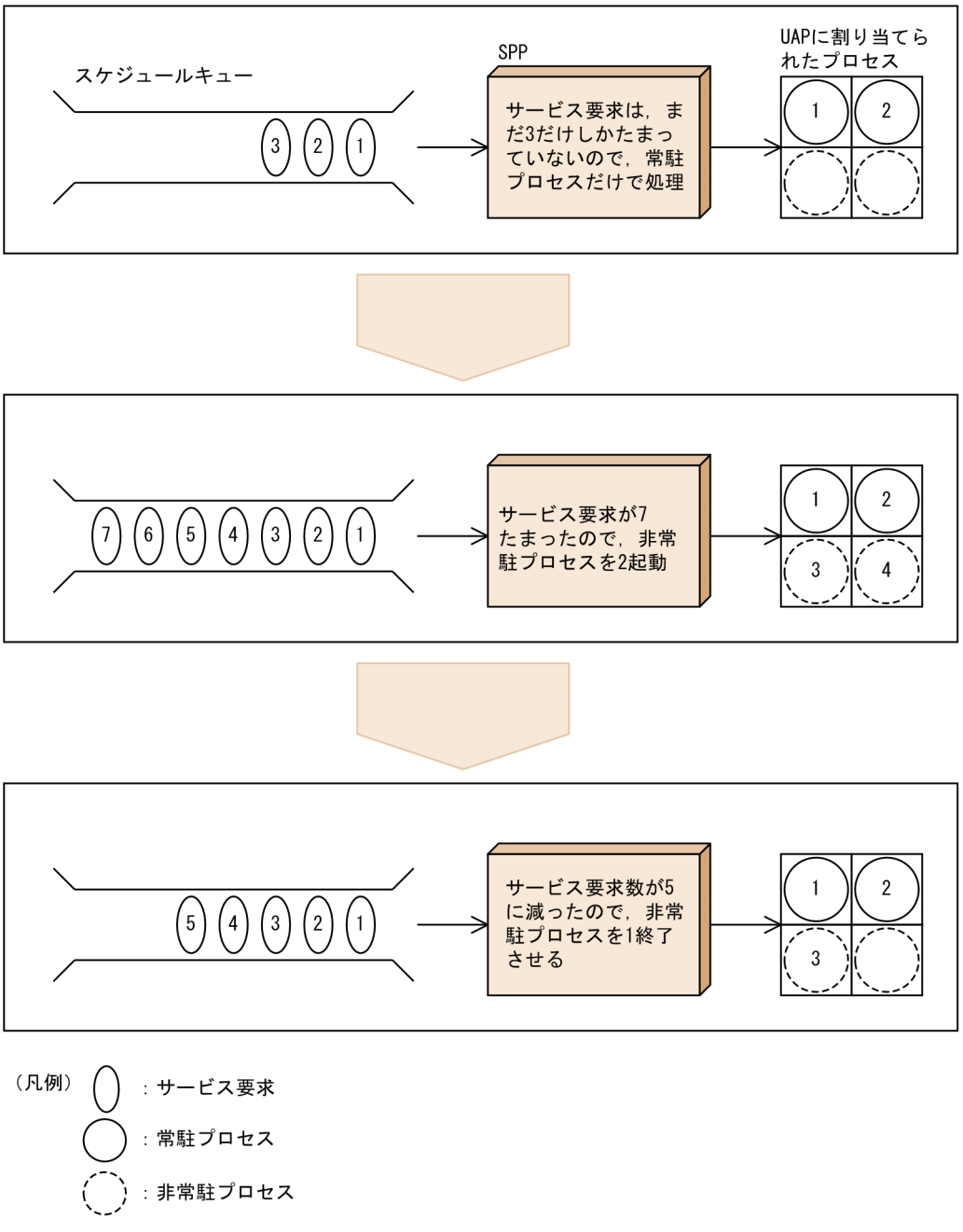
一つ一つのユーザサーバには，**スケジュールの優先度**（スケジュールプライオリティ）を付けられます。優先度が高いユーザサーバの非常駐プロセスは，ほかの非常駐プロセスに比べて優先的にスケジュールされます。

スケジュールの優先度は，ユーザサーバで使うプロセスを設定するときに，一緒に設定します。

プロセスの負荷分散を次の図に示します。

図 1-26　プロセスの負荷分散

●常駐プロセス：2，非常駐プロセス：2，balance_countの値：2の例



(7) ノード間負荷バランス機能

同じサービスグループ名のユーザサーバを複数のノードに配置しておくと，サービスを要求されたときに，どのノードのユーザサーバでも処理できます。これによって，ノード間で負荷分散できます。これを**ノード間負荷バランス機能**といいます。ノード間負荷バランス機能を使うための環境設定は必要ありません。複数のノードで同じサービスグループ名のユーザサーバを開始しておけば，OpenTP1 で自動的に振り分けます。

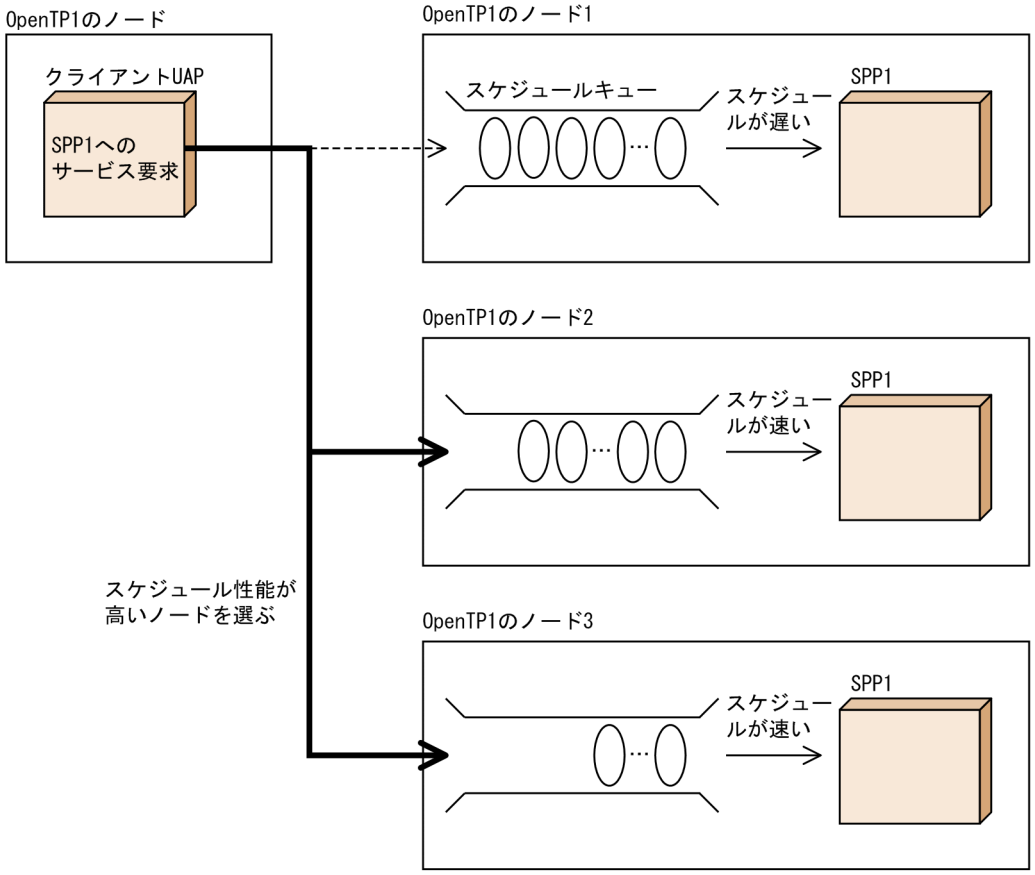
OpenTP1 システム内の同一サービスグループに，マルチスケジューラ機能を使用しているユーザサーバと，マルチスケジューラ機能を使用していないユーザサーバが混在する場合，マルチスケジューラ機能を使用しているユーザサーバが高負荷状態でも，マルチスケジューラ機能を使用していないユーザサーバには負荷分散されません。マルチスケジューラ機能を使用していないユーザサーバに負荷分散するには，スケジュールサービス定義の scdmulti 定義コマンドに-t オプションを指定してください。scdmulti 定義コマンドの詳細については，マニュアル「OpenTP1 システム定義」のスケジュールサービス定義を参照してください。

ノード間負荷バランス機能で負荷を分散できるノードの数は，最大 128 です。

ノード間負荷バランス機能では，ノードのスケジュール状態に応じて，より効率的に処理できるノードへ負荷を分散させます。ノード間負荷バランス機能の環境で，サービスを要求した UAP と同じノードにあるユーザサーバを優先的にスケジュールしたい場合には，そのノードのスケジュールサービス定義に scd_this_node_first オペランドに Y を指定しておいてください。

ノード間負荷バランス機能の概要を次の図に示します。

図 1-27　ノード間負荷バランス機能の概要



(8)　ノード間負荷バランス拡張機能

ユーザは次に示す指定ができます。

- LEVEL0 のノードへのスケジュール比率の指定

  スケジュールサービス定義の schedule_rate オペランドを指定することによって，負荷レベルが LEVEL0 のノードにスケジュールする比率（%）を指定できます。

- 負荷監視インターバル時間の指定

  ユーザサービス定義およびユーザサービスデフォルト定義の loadcheck_interval オペランドを指定することによって，サービスグループごとに負荷監視インターバル時間を指定できます。

- 負荷レベルのしきい値の指定

  ユーザサービス定義およびユーザサービスデフォルト定義の levelup_queue_count オペランドおよび leveldown_queue_count オペランドを指定することによって，サービスグループごとにサービス要求滞留数によって負荷レベルを決定するしきい値を指定できます。

- 通信障害時のリトライ回数の指定

  通常，サービス要求のスケジューリング時に通信障害が発生すると，再スケジュールしないでエラーリターンします。スケジュールサービス定義の scd_retry_of_comm_error オペランドを指定することによって，通信障害が発生したノード以外へスケジュールするリトライ回数を指定できます。

なお，この機能は，TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証できませんので，ご了承ください。

(9)　マルチスケジューラ機能

クライアント UAP から，他ノードのキュー受信型サーバ（スケジュールキューを使う SPP）にサービスを要求した場合，要求先サーバが存在するノードのスケジューラデーモンが，いったんサービス要求メッセージを受信し，該当するキュー受信型サーバのスケジュールキューに格納します。スケジューラデーモンは，スケジュールサービスを提供するシステムデーモンのことです。

スケジューラデーモンは，OpenTP1 システムごとに 1 プロセスです。そのため，システムの大規模化，マシンやネットワークの高性能化などに伴って，従来のスケジューラデーモンだけでは効率良くスケジューリングできないことがあります。従来のスケジューラデーモンだけでは効率良くスケジューリングできない場合については，「付録 C　マルチスケジューラ機能の検討が必要なシステム構成例」を参照してください。

従来のスケジューラデーモン（これ以降**マスタスケジューラデーモン**といいます）とは別に，サービス要求受信専用デーモン（これ以降**マルチスケジューラデーモン**といいます）を複数プロセス起動し，サービス要求メッセージ受信処理を並行動作させることによって，受信処理の競合によるスケジューリング遅延を回避できます。この機能を**マルチスケジューラ機能**といいます。

なお，この機能は，TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証できませんので，ご了承ください。

マルチスケジューラ機能を使用するには，次の定義を指定する必要があります。

RPC 受信側

スケジュールサービス定義　scdmulti

ユーザサービス定義　scdmulti

RPC 送信側

ユーザサービス定義　multi_schedule

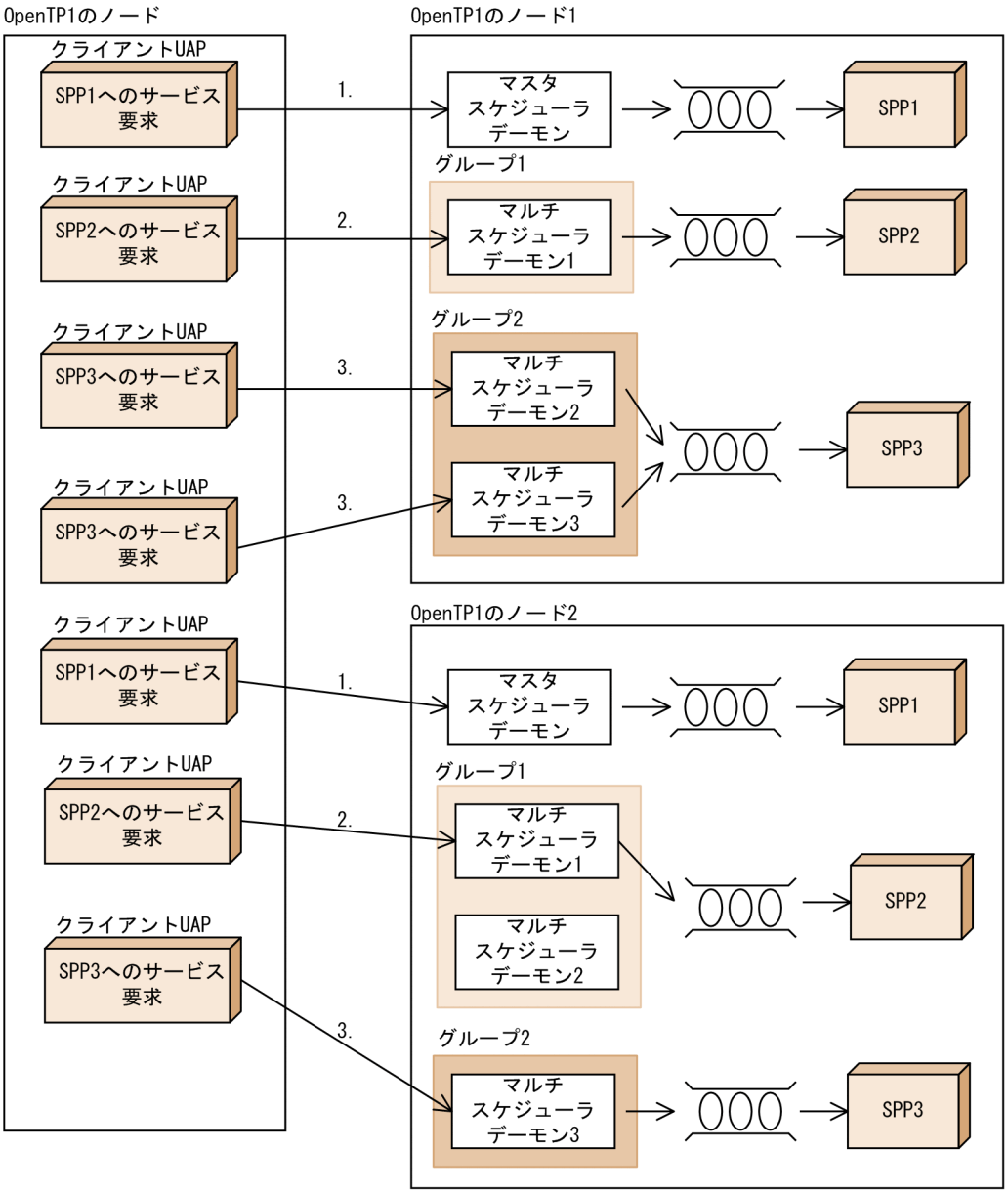
また，幾つかのマルチスケジューラデーモンをキュー受信型サーバごとにグループ化できます。これによって，異なるサーバ間でサービス要求メッセージの受信処理が競合するのを回避できます。マルチスケジューラデーモンをグループ化して使用する場合，サーバ側でユーザサービス定義 scdmulti に-g オプションを指定する必要があります。

OpenTP1 起動時に，スケジュールサービスを提供するシステムデーモンとして，マスタスケジューラデーモンに加え，定義に指定したマルチスケジューラデーモンをウェルノウンポート番号で起動します。なお，TP1/Client からマルチスケジューラ機能を使用してサービスを要求する場合については，マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」を参照してください。

マルチスケジューラ機能を使用した RPC については「2.1.16　マルチスケジューラ機能を使用した RPC」を参照してください。

マルチスケジューラ機能の使用例を次の図に示します。

図 1-28　マルチスケジューラ機能の使用例



- SPP1 は，短いサービス要求メッセージを扱うため，マルチスケジューラ機能を使用しないで，マスタスケジューラデーモンにスケジューリングさせます（図の 1.）。
- SPP2 は，長大サービス要求メッセージを扱うため，OpenTP1 のノード 1 とノード 2 に分散させ，ノード 1 にあるグループ 1 のマルチスケジューラデーモン 1，またはノード 2 にあるグループ 1 のマルチスケジューラデーモン 1，2 にスケジューリングさせます（図の 2.）。
- SPP3 は，短いサービス要求メッセージを扱いますが，サービス要求数が多いため，OpenTP1 のノード 1 とノード 2 に分散させ，ノード 1 にあるグループ 2 のマルチスケジューラデーモン 2，3，またはノード 2 にあるグループ 2 のマルチスケジューラデーモン 3 にスケジューリングさせます（図の 3.）。

1.4 OpenTP1 のライブラリ関数

1.4.1 アプリケーションプログラミングインタフェースの機能

OpenTP1 のライブラリ関数を使ってできる機能を，次に示します。

(1) OpenTP1 の基本機能（TP1/Server Base，TP1/LiNK）

- リモートプロシジャコール
  UAP 間で，関数呼び出しと同様の方法で通信できます。
- トランザクション制御
  UAP の処理をトランザクションとして制御できます。
- システム運用の管理
  UAP からコマンドを実行したり，ユーザサーバの状態を報告したりできます。
- メッセージログの出力
  UAP からユーザ任意の情報を，メッセージログとして出力できます。
- ユーザジャーナルの取得
  ユーザジャーナル（UJ）を，システムジャーナルファイルに取得できます。
- ジャーナルデータの編集
  jnlrput コマンドの実行結果を格納したファイルにあるジャーナルデータを，編集できます。
- リアルタイム統計情報の取得
  UAP 内の任意区間で，リアルタイム統計情報を取得できます。

(2) TP1/Message Control を使う場合の機能

- メッセージ送受信
  広域網，および TCP/IP で接続されたシステム間で，メッセージ送受信形態の通信ができます。

(3) ユーザデータを使う場合の機能

- DAM ファイルサービス（TP1/FS/Direct Access）
  OpenTP1 専用のユーザファイルを，直接編成ファイル（DAM ファイル）として使えるようにします。
- TAM ファイルサービス（TP1/FS/Table Access）
  OpenTP1 専用のユーザファイルを，テーブルアクセス法による直接編成ファイル（TAM ファイル）として使えるようにします。
- ISAM ファイルサービス※（ISAM，ISAM/B）
  X/Open の ISAM モデルに準拠した，索引順編成ファイル（ISAM ファイル）を使えるようにします。

- IST サービス（TP1/Shared Table Access）
  共用メモリ上にあるテーブル（IST テーブル）を，複数の OpenTP1 システムで共用できるようにします。IST サービスの場合，ユーザファイルの実体はなく，メモリ上の IST テーブルにデータが格納されます。
- 資源の排他制御
  任意のファイル（UNIX ファイル）を，OpenTP1 の API で排他制御します。

注※
ISAM ファイルサービスについては，マニュアル「索引順編成ファイル管理 ISAM」を参照してください。

(4) X/Open に準拠したアプリケーションプログラミングインタフェース

- XATMI インタフェース
  X/Open に準拠した API で，クライアント／サーバ形態の通信ができます。
- TX インタフェース
  X/Open に準拠した API で，トランザクションを制御できます。

(5) 特殊な形態で使う機能

特殊な形態で使う機能の関数を，次に示します。

(a) TP1/Multi を使う場合の機能

- マルチノード機能
  クラスタ／並列システム形態の OpenTP1 の UAP で，各種の機能を使えます。

(b) オンラインテスタ（TP1/Online Tester）を使う場合の機能

- オンラインテスタの管理
  ユーザサーバのテスト状態を，UAP から関数を使って知ることができます。

1.4.2 OpenTP1 のライブラリ関数の一覧

(1) ライブラリ関数の一覧

OpenTP1 のライブラリ関数の一覧を表 1-1～表 1-5 に示します。

表 1-1　OpenTP1 のライブラリ関数の一覧（OpenTP1 の基本機能の関数）

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
リモートプロシジャコール	アプリケーションプログラムの開始	dc_rpc_open	CBLDCRPC('OPEN ')
	SPP のサービス開始	dc_rpc_mainloop	CBLDCRSV('MAINLOOP')
	遠隔サービスの要求	dc_rpc_call	CBLDCRPC('CALL ')
	通信先を指定した遠隔サービスの呼び出し※1	dc_rpc_call_to	－
	処理結果の非同期受信	dc_rpc_poll_any_replies	CBLDCRPC('POLLANYR')
	エラーが発生した非同期応答型 RPC 要求の記述子の取得	dc_rpc_get_error_descriptor	CBLDCRPC('GETERDES')
	処理結果の受信の拒否	dc_rpc_discard_further_replies	CBLDCRPC('DISCARDF')
	特定の処理結果の受信の拒否	dc_rpc_discard_specific_reply	CBLDCRPC('DISCARDS')
	サービス関数のリトライ	dc_rpc_service_retry	CBLDCRPC('SVRETRY ')
	サービス要求のスケジュールプライオリティの設定	dc_rpc_set_service_prio	CBLDCRPC('SETSVPRI')
	サービス要求のスケジュールプライオリティの参照	dc_rpc_get_service_prio	CBLDCRPC('GETSVPRI')
	サービスの応答待ち時間の参照	dc_rpc_get_watch_time	CBLDCRPC('GETWATCH')
	サービスの応答待ち時間の更新	dc_rpc_set_watch_time	CBLDCRPC('SETWATCH')
	クライアント UAP のノードアドレスの取得	dc_rpc_get_callers_address	CBLDCRPC('GETCLADR')
	ゲートウェイのノードアドレスの取得	dc_rpc_get_gateway_address	CBLDCRPC('GETGWADR')
	CUP への一方通知	dc_rpc_cltsend	CBLDCRPC('CLTSEND ')
	アプリケーションプログラムの終了	dc_rpc_close	CBLDCRPC('CLOSE ')
リモート API 機能	rap リスナーとのコネクション確立	dc_rap_connect	CBLDCRAP('CONNECT ') CBLDCRAP('CONNECTX')
	rap リスナーとのコネクション解放	dc_rap_disconnect	CBLDCRAP('DISCNCT ')

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
トランザクション制御	トランザクションの開始	dc_trn_begin	CBLDCTRN('BEGIN ')
	連鎖モードのコミット	dc_trn_chained_commit	CBLDCTRN('C-COMMIT')
	連鎖モードのロールバック	dc_trn_chained_rollback	CBLDCTRN('C-ROLL ')
	非連鎖モードのコミット	dc_trn_unchained_commit	CBLDCTRN('U-COMMIT')
	非連鎖モードのロールバック	dc_trn_unchained_rollback	CBLDCTRN('U-ROLL ')
	現在のトランザクションに関する情報の報告	dc_trn_info	CBLDCTRN('INFO ')
	リソースマネジャ接続先選択	dc_trn_rm_select	CBLDCTRN('RMSELECT')
システム運用の管理	運用コマンドの実行	dc_adm_call_command	CBLDCADM('COMMAND ')
	ユーザサーバの開始処理完了の報告	dc_adm_complete	CBLDCADM('COMPLETE')
	ユーザサーバの状態の報告	dc_adm_status	CBLDCADM('STATUS ')
監査ログの出力	監査ログの出力	dc_log_audit_print	CBLDCADT('PRINT ')
メッセージログの出力	メッセージログの出力	dc_logprint	CBLDCLOG('PRINT ')
ユーザジャーナルの取得	ユーザジャーナルの取得	dc_jnl_ujput	CBLDCJNL('UJPUT ')
ジャーナルデータの編集※2	jnlrput 出力ファイルのクローズ	－	CBLDCJUP('CLOSERPT')
	jnlrput 出力ファイルのオープン	－	CBLDCJUP('OPENRPT ')
	jnlrput 出力ファイルからジャーナルデータの入力	－	CBLDCJUP('RDGETRPT')
性能検証用トレース	ユーザ固有の性能検証用トレースの取得	dc_prf_utrace_put	CBLDCPRF('PRFPUT ')
	性能検証用トレース取得通番の通知	dc_prf_get_trace_num	CBLDCPRF('PRFGETN ')
リアルタイム統計情報サービス	任意区間でのリアルタイム統計情報の取得	dc_rts_utrace_put	CBLDCRTS('RTSPUT ')

(凡例)

－：該当しません。

注※1

COBOL-UAP 作成用プログラムは使えません。

注※2

ジャーナルデータの編集では，C 言語の API は使えません。

表 1-2　OpenTP1 のライブラリ関数の一覧（TP1/Message Control の関数）

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
メッセージ送受信	MCF 環境のオープン	dc_mcf_open	CBLDCMCF('OPEN ')
	MHP のサービス開始	dc_mcf_mainloop	CBLDCMCF('MAINLOOP')
	メッセージの受信	dc_mcf_receive	CBLDCMCF('RECEIVE ')
	応答メッセージの送信	dc_mcf_reply	CBLDCMCF('REPLY ')
	メッセージの送信	dc_mcf_send	CBLDCMCF('SEND ')
	メッセージの再送	dc_mcf_resend	CBLDCMCF('RESEND ')
	同期型のメッセージ受信	dc_mcf_recvsync	CBLDCMCF('RECVSYNC')
	同期型のメッセージ送信	dc_mcf_sendsync	CBLDCMCF('SENDSYNC')
	同期型のメッセージ送受信	dc_mcf_sendrecv	CBLDCMCF('SENDRECV')
	一時記憶データの受け取り	dc_mcf_tempget	CBLDCMCF('TEMPGET ')
	一時記憶データの更新	dc_mcf_tempput	CBLDCMCF('TEMPPUT ')
	継続問い合わせ応答の終了	dc_mcf_contend	CBLDCMCF('CONTEND ')
	アプリケーションプログラムの起動	dc_mcf_execap	CBLDCMCF('EXECAP ')
	アプリケーション情報通知	dc_mcf_ap_info	CBLDCMCF('APINFO ')
	UOC アプリケーション情報通知	dc_mcf_ap_info_uoc	－
	ユーザタイマ監視の設定	dc_mcf_timer_set	CBLDCMCF('TIMERSET')
	ユーザタイマ監視の取り消し	dc_mcf_timer_cancel	CBLDCMCF('TIMERCAN')
	MHP のコミット	dc_mcf_commit	CBLDCMCF('COMMIT ')
	MHP のロールバック	dc_mcf_rollback	CBLDCMCF('ROLLBACK')
	MCF 環境のクローズ	dc_mcf_close	CBLDCMCF('CLOSE ')
	MCF 通信サービスの状態取得	dc_mcf_tlscom	CBLDCMCF('TLSCOM ')
	コネクションの状態取得	dc_mcf_tlscn	CBLDCMCF('TLSCN ')
	コネクションの確立	dc_mcf_tactcn	CBLDCMCF('TACTCN ')
	コネクションの解放	dc_mcf_tdctcn	CBLDCMCF('TDCTCN ')
	サーバ型コネクションの確立要求の受付状態取得	dc_mcf_tlsln	CBLDCMCF('TLSLN ')
	サーバ型コネクションの確立要求の受付開始	dc_mcf_tonln	CBLDCMCF('TONLN ')

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
メッセージ送受信	サーバ型コネクションの確立要求の受付終了	dc_mcf_tofln	CBLDCMCF('TOFLN ')
	アプリケーションに関するタイマ起動要求の削除	dc_mcf_adltap	CBLDCMCF('ADLTAP ')
	論理端末の状態取得	dc_mcf_tlsle	CBLDCMCF('TLSLE ')
	論理端末の閉塞	dc_mcf_tdctle	CBLDCMCF('TDCTLE ')
	論理端末の閉塞解除	dc_mcf_tactle	CBLDCMCF('TACTLE ')
	論理端末の出力キュー削除	dc_mcf_tdlqle	CBLDCMCF('TDLQLE ')

(凡例)

－：該当しません。

表 1-3　OpenTP1 のライブラリ関数の一覧（ユーザデータを操作する関数）

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
DAM ファイルサービス	論理ファイルのオープン	dc_dam_open	CBLDCDAM('DCDAMSVC','OPEN')
	論理ファイルからブロックの入力	dc_dam_read	CBLDCDAM('DCDAMSVC','READ')
	論理ファイルのブロックの更新	dc_dam_rewrite	CBLDCDAM('DCDAMSVC','REWT')
	論理ファイルへブロックの出力	dc_dam_write	CBLDCDAM('DCDAMSVC','WRIT')
	論理ファイルのクローズ	dc_dam_close	CBLDCDAM('DCDAMSVC','CLOS')
	論理ファイルの閉塞	dc_dam_hold	CBLDCDAM('DCDAMSVC','HOLD')
	論理ファイルの閉塞の解除	dc_dam_release	CBLDCDAM('DCDAMSVC','RLES')
	論理ファイルの状態の参照	dc_dam_status	CBLDCDAM('DCDAMSVC','STAT')
	回復対象外 DAM ファイル使用の開始	dc_dam_start	CBLDCDAM('DCDAMSVC','STRT')
	回復対象外 DAM ファイル使用の終了	dc_dam_end	CBLDCDAM('DCDAMSVC','END ')
	物理ファイルの割り当て	dc_dam_create	CBLDCDMB('DCDAMINT','CRAT')
	物理ファイルのオープン	dc_dam_iopen	CBLDCDMB('DCDAMINT','OPEN')
	物理ファイルからブロックの入力	dc_dam_get	CBLDCDMB('DCDAMINT','GET ')
	物理ファイルへブロックの出力	dc_dam_put	CBLDCDMB('DCDAMINT','PUT ')
	物理ファイルのブロックの検索	dc_dam_bseek	CBLDCDMB('DCDAMINT','BSEK')

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
DAM ファイルサービス	物理ファイルからブロックの直接入力	dc_dam_dget	CBLDCDMB('DCDAMINT','DGET')
	物理ファイルへブロックの直接出力	dc_dam_dput	CBLDCDMB('DCDAMINT','DPUT')
	物理ファイルのクローズ	dc_dam_iclose	CBLDCDMB('DCDAMINT','CLOS')
TAM ファイルサービス	TAM テーブルのオープン※	dc_tam_open	－
	TAM テーブルからレコードの入力	dc_tam_read	CBLDCTAM('FxxR')('FxxU')('VxxR')('VxxU')
	TAM テーブルのレコード入力を前提の更新	dc_tam_rewrite	CBLDCTAM('MFY ')('MFYS')('STR ')('WFY ') ('WFYS')('YTR ')
	TAM テーブルのレコードの更新／追加	dc_tam_write	
	TAM テーブルのレコードの削除	dc_tam_delete	CBLDCTAM('ERS ')('ERSR')('BRS ')('BRSR')
	TAM テーブルのレコードの入力取り消し※	dc_tam_read_cancel	－
	TAM テーブルの状態の取得	dc_tam_get_inf	CBLDCTAM('GST ')
	TAM テーブルの情報の取得	dc_tam_status	CBLDCTAM('INFO')
	TAM テーブルのクローズ※	dc_tam_close	－
IST サービス	IST テーブルのオープン	dc_ist_open	CBLDCIST('DCISTSVC','OPEN')
	IST テーブルからレコードの入力	dc_ist_read	CBLDCIST('DCISTSVC','READ')
	IST テーブルへレコードの出力	dc_ist_write	CBLDCIST('DCISTSVC','WRIT')
	IST テーブルのクローズ	dc_ist_close	CBLDCIST('DCISTSVC','CLOS')
資源の排他制御	資源の排他	dc_lck_get	CBLDCLCK('GET ')
	全資源の排他の解除	dc_lck_release_all	CBLDCLCK('RELALL ')
	資源名称を指定した排他の解除	dc_lck_release_byname	CBLDCLCK('RELNAME ')

(凡例)

－：該当しません。

注※

COBOL-UAP 作成用プログラムは使えません。

表 1-4　OpenTP1 のライブラリ関数の一覧（X/Open に準拠した関数）

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
XATMI インタフェース	リクエスト／レスポンス型サービスの呼び出しと応答の受信	tpcall()	TPCALL
	リクエスト／レスポンス型サービスの呼び出し	tpacall()	TPACALL
	リクエスト／レスポンス型サービスからの非同期応答の受信	tpgetrply()	TPGETRPLY
	リクエスト／レスポンス型サービスのキャンセル	tpcancel()	TPCANCEL
	会話型サービスとのコネクションの確立	tpconnect()	TPCONNECT
	会話型サービスとのコネクションの切断	tpdiscon()	TPDISCON
	会話型サービスからのメッセージの受信	tprecv()	TPRECV
	会話型サービスへのメッセージの送信	tpsend()	TPSEND
	型付きバッファの割り当て	tpalloc()	－
	型付きバッファの解放	tpfree()	－
	型付きバッファのサイズの変更	tprealloc()	－
	型付きバッファの情報の取得	tptypes()	－
	サービス名の広告	tpadvertise()	TPADVERTISE
	サービス名の広告の取り消し	tpunadvertise()	TPUNADVERTISE
	サービス関数のテンプレート	tpservice()	TPSVCSTART
	サービス関数からのリターン	tpreturn()	TPRETURN
TX インタフェース	トランザクションの開始	tx_begin()	TXBEGIN
	トランザクションのコミット	tx_commit()	TXCOMMIT
	現在のトランザクションに関する情報の返却	tx_info()	TXINFORM
	リソースマネジャ集合のオープン	tx_open()	TXOPEN
	トランザクションのロールバック	tx_rollback()	TXROLLBACK
	リソースマネジャ集合のクローズ	tx_close()	TXCLOSE

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
TX インタフェース	commit_return 特性の設定	tx_set_commit_return()	TXSETCOMMITRET
	transaction_control 特性の設定	tx_set_transaction_control()	TXSETTRANCTL
	transaction_timeout 特性の設定	tx_set_transaction_timeout()	TXSETTIMEOUT

（凡例）

－：この機能に該当する XATMI インタフェースの COBOL の API はありません。

表 1-5　OpenTP1 のライブラリ関数の一覧（特殊な形態で使う関数）

機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL-UAP 作成用プログラム
マルチノード機能※	OpenTP1 ノードのステータス取得の開始	dc_adm_get_nd_status_begin	－
	OpenTP1 ノードのステータスの取得	dc_adm_get_nd_status_next	－
	指定した OpenTP1 ノードのステータスの取得	dc_adm_get_nd_status	－
	OpenTP1 ノードのステータス取得の終了	dc_adm_get_nd_status_done	－
	ノード識別子の取得の開始	dc_adm_get_nodeconf_begin	－
	ノード識別子の取得	dc_adm_get_nodeconf_next	－
	ノード識別子の取得の終了	dc_adm_get_nodeconf_done	－
	指定したノード識別子の取得	dc_adm_get_node_id	－
	ユーザサーバのステータス取得の開始	dc_adm_get_sv_status_begin	－
	ユーザサーバのステータスの取得	dc_adm_get_sv_status_next	－
	指定したユーザサーバのステータスの取得	dc_adm_get_sv_status	－
	ユーザサーバのステータス取得の終了	dc_adm_get_sv_status_done	－
オンラインテスタの管理	ユーザサーバのテスト状態の報告	dc_uto_test_status	CBLDCUTO('T-STATUS')

（凡例）

－：該当しません。

注※

マルチノード機能では，COBOL-UAP 作成用プログラムは使えません。

(2) アプリケーションプログラムで使えるライブラリ関数

OpenTP1 の UAP で使えるライブラリ関数を表 1-6～表 1-10 に示します。ここでは，SUP，SPP，MHP，および**オフラインの業務をする UAP** について示します。

表 1-6 UAP で使えるライブラリ関数（OpenTP1 の基本機能の関数）

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をする UAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
dc_rpc_open	○	−	○M	−	−	○M	−	−
dc_rpc_mainloop	−	−	○M	−	−	−	−	−
dc_rpc_call	○	○	○	○	○	○	○	−
dc_rpc_call_to	○	○	○	○	○	○	○	−
dc_rpc_poll_any_replies	○	○	○	○	○	○	○	−
dc_rpc_get_error_descriptor	○	○	○	○	○	○	○	−
dc_rpc_discard_further_replies	○	○	○	○	○	○	○	−
dc_rpc_discard_specific_reply	○	○	○	○	○	○	○	−
dc_rpc_service_retry	−	−	○	−	−	○	−	−
dc_rpc_set_service_prio	○	○	○	○	○	○	○	−
dc_rpc_get_service_prio	○	○	○	○	○	○	○	−
dc_rpc_get_watch_time	○	○	○	○	○	○	○	−
dc_rpc_set_watch_time	○	○	○	○	○	○	○	−
dc_rpc_get_callers_address	−	−	○	○	○	−	−	−
dc_rpc_get_gateway_address	−	−	○	○	○	−	−	−
dc_rpc_cltsend	○	○	○	○	○	○	○	−
dc_rpc_close	○	−	○M	−	−	○M	−	−
dc_rap_connect	○	−	○	−	−	○	−	−
dc_rap_disconnect	○	−	○	−	−	○	−	−

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をするUAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
dc_trn_begin※	○	−	○	−	−	$○_M$	−	−
dc_trn_chained_commit※	−	○	−	○	−	−	−	−
dc_trn_chained_rollback※	−	○	−	○	−	−	−	−
dc_trn_unchained_commit※	−	○	−	○	−	−	$○_M$	−
dc_trn_unchained_rollback※	−	○	−	○	○	−	$○_M$	−
dc_trn_info	○	○	○	○	○	○	○	−
dc_trn_rm_select	○	−	○	−	−	−	−	−
dc_adm_call_command	○	○	○	○	○	○	○	−
dc_adm_complete	○	−	−	−	−	−	−	−
dc_adm_status	○	○	○	○	○	○	○	−
dc_log_audit_print	○	○	○	○	○	○	○	−
dc_logprint	○	○	○	○	○	○	○	−
dc_jnl_ujput※	−	○	−	○	○	−	○	−
CBLDCJUP('CLOSERPT')	−	−	−	−	−	−	−	○
CBLDCJUP('OPENRPT ')	−	−	−	−	−	−	−	○
CBLDCJUP('RDGETRPT')	−	−	−	−	−	−	−	○
dc_prf_utrace_put	○	○	○	○	○	○	○	○
dc_prf_get_trace_num	○	○	○	○	○	○	○	○
dc_rts_utrace_put	○	○	○	○	○	○	○	−

（凡例）

○：該当する条件で使えます。

$○_M$：メイン関数からだけ，関数を使えます。

−：該当する条件では使えません。

注

MHP の「トランザクション処理の範囲でない」とは，非トランザクション属性の MHP，または MHP のメイン関数の範囲を示します。

注※

この関数を使う UAP は，トランザクションとして実行する指定をしてください。

・TP1/Server Base の場合：ユーザサービス定義で atomic_update オペランドに Y を指定

・TP1/LiNK の場合：アプリケーションプログラムの環境を設定するときに，トランザクション機能を使う指定

表 1-7　UAP で使えるライブラリ関数（TP1/Message Control の関数）

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をする UAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
dc_mcf_open	−	−	○M	−	−	○M	○M	−
dc_mcf_mainloop	−	−	−	−	−	○M	−	−
dc_mcf_receive	−	−	−	−	−	○NO	○	−
dc_mcf_reply	−	−	−	−	−	○NO	○	−
dc_mcf_send	−	−	−	○	○	○NO	○	−
dc_mcf_resend	−	−	−	○	○	−	○	−
dc_mcf_recvsync	−	−	○	○	○	○	○	−
dc_mcf_sendsync	−	−	○	○	○	○	○	−
dc_mcf_sendrecv	−	−	○	○	○	○	○	−
dc_mcf_tempget	−	−	−	−	−	○NO	○	−
dc_mcf_tempput	−	−	−	−	−	○NO	○	−
dc_mcf_contend	−	−	−	−	−	○NO	○	−
dc_mcf_execap	−	−	−	○	○	○NO	○	−
dc_mcf_ap_info	−	−	−	−	−	○NO	○	−
dc_mcf_ap_info_uoc	−	−	−	−	−	○NO	○	−
dc_mcf_timer_set	−	−	○	○	○	○	○	−
dc_mcf_timer_cancel	−	−	○	○	○	○	○	−
dc_mcf_commit	−	−	−	−	−	−	○	−
dc_mcf_rollback	−	−	−	−	−	−	○	−
dc_mcf_close	−	−	○M	−	−	○M	○M	−
dc_mcf_tlscom	−	−	○	○	○	○	○	−
dc_mcf_tlscn	−	−	○	○	○	○	○	−
dc_mcf_tactcn	−	−	○	○	○	○	○	−
dc_mcf_tdctcn	−	−	○	○	○	○	○	−
dc_mcf_tlsln	−	−	○	○	○	○	○	−

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をするUAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
dc_mcf_tonln	−	−	○	○	○	○	○	−
dc_mcf_tofln	−	−	○	○	○	○	○	−
dc_mcf_adltap	−	−	○	○	○	○	○	−
dc_mcf_tlsle	−	−	○	○	○	○	○	−
dc_mcf_tdctle	−	−	○	○	○	○	○	−
dc_mcf_tactle	−	−	○	○	○	○	○	−
dc_mcf_tdlqle	−	−	○	○	○	○	○	−

（凡例）

○：該当する条件で使えます。

$○_{M}$：メイン関数からだけ，関数を使えます。

$○_{NO}$：非トランザクション属性の MHP のサービス関数の範囲でだけ，関数を使えます。

−：該当する条件では使えません。

注

MHP の「トランザクション処理の範囲でない」とは，非トランザクション属性の MHP，または MHP のメイン関数の範囲を示します。

表 1-8　UAP で使えるライブラリ関数（ユーザデータを操作する関数）

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をするUAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
dc_dam_open※	○	○	○	○	○	○	○	−
dc_dam_read※	○	○	○	○	○	○	○	−
dc_dam_rewrite※	(○)	○	(○)	○	○	(○)	○	−
dc_dam_write※	(○)	○	(○)	○	○	(○)	○	−
dc_dam_close※	○	○	○	○	○	○	○	−
dc_dam_hold	○	○	○	○	○	−	○	−
dc_dam_release	○	○	○	○	○	−	○	−
dc_dam_status	○	○	○	○	○	○	○	−

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をするUAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
dc_dam_start	○	○	○	○	○	○	○	−
dc_dam_end	○	○	○	○	○	○	○	−
dc_dam_create	−	−	−	−	−	−	−	○
dc_dam_iopen	−	−	−	−	−	−	−	○
dc_dam_get	−	−	−	−	−	−	−	○
dc_dam_put	−	−	−	−	−	−	−	○
dc_dam_bseek	−	−	−	−	−	−	−	○
dc_dam_dget	−	−	−	−	−	−	−	○
dc_dam_dput	−	−	−	−	−	−	−	○
dc_dam_iclose	−	−	−	−	−	−	−	○
dc_tam_open	○	○	○	○	○	○	○	−
dc_tam_read	−	○	−	○	○	−	○	−
dc_tam_rewrite	−	○	−	○	○	−	○	−
dc_tam_write	−	○	−	○	○	−	○	−
dc_tam_delete	−	○	−	○	○	−	○	−
dc_tam_read_cancel	−	○	−	○	○	−	○	−
dc_tam_get_inf	○	○	○	○	○	○	○	−
dc_tam_status	○	○	○	○	○	○	○	−
dc_tam_close	○	○	○	○	○	○	○	−
dc_ist_open	○	○	○	○	○	○	○	−
dc_ist_read	○	○	○	○	○	○	○	−
dc_ist_write	○	○	○	○	○	○	○	−
dc_ist_close	○	○	○	○	○	○	○	−
dc_lck_get※	−	○	−	○	○	−	○	−
dc_lck_release_all※	−	○	−	○	○	−	○	−
dc_lck_release_byname※	−	○	−	○	○	−	○	−

（凡例）

○：該当する条件で使えます。

（○）：回復対象外の DAM ファイルのときだけ使えます。

－：該当する条件では使えません。

注

MHP の「トランザクション処理の範囲でない」とは，非トランザクション属性の MHP，または MHP のメイン関数の範囲を示します。

注※

この関数を使う UAP は，TP1/Server Base の場合，トランザクション属性の指定（ユーザサービス定義で atomic_update オペランドに Y を指定）してください。ただし，回復対象外の DAM ファイルへアクセスする場合はトランザクション処理を前提としません。

TP1/LiNK の UAP では，これらの関数は使えません。

表 1-9　UAP で使えるライブラリ関数（X/Open に準拠した関数）

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をする UAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
tpcall	○	○	○	○	○	－	－	－
tpacall	○	○	○	○	○	－	－	－
tpgetrply	○	○	○	○	○	－	－	－
tpcancel	○	○	○	○	○	－	－	－
tpconnect	○	○	○	○	○	－	－	－
tpdiscon	○	○	○	○	○	－	－	－
tprecv	○	○	○	○	○	－	－	－
tpsend	○	○	○	○	○	－	－	－
tpalloc	○	○	○	○	○	－	－	－
tpfree	○	○	○	○	○	－	－	－
tprealloc	○	○	○	○	○	－	－	－
tptypes	○	○	○	○	○	－	－	－
tpadvertise	－	－	○※1	○※1	○※1	－	－	－
tpunadvertise	－	－	○※1	○※1	○※1	－	－	－
tpservice※2	－	－	－	－	－	－	－	－
tpreturn	－	－	○※3	○※3	○※3	－	－	－
tx_begin※4	○	－	○	－	－	○	－	－

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をするUAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
tx_commit TX_CHAINED 指定※4	−	○	○	−	−	−	−	−
tx_commit TX_UNCHAINED 指定※4	−	○	○	−	−	−	−	−
tx_info	○	○	○	○	○	−	−	−
tx_open	○	−	○	−	−	−	−	−
tx_rollback TX_CHAINED 指定※4	−	○	−	○	−	−	−	−
tx_rollback TX_UNCHAINED 指定※4	−	○	−	○	○	−	−	−
tx_close	○	−	○	−	−	−	−	−
tx_set_commit_return※4	○	○	○	○	○	−	−	−
tx_set_transaction _control※4	○	○	○	○	○	−	−	−
tx_set_transaction _timeout※4	○	○	○	○	○	−	−	−

（凡例）

○：該当する条件で使えます。

−：該当する条件では使えません。

注※1

この関数は，サービス関数の中でだけ使えます。

注※2

tpservice は，サービス関数の実体です。

注※3

この関数は，XATMI インタフェースのサービス関数をリターンするためだけに使います。

注※4

この関数を使う UAP は，トランザクションとして実行する指定をしてください。

・TP1/Server Base の場合：ユーザサービス定義で atomic_update オペランドに Y を指定

・TP1/LiNK の場合：アプリケーションプログラムの環境を設定するときに，トランザクション機能を使う指定

表 1-10　UAP で使えるライブラリ関数（特殊な形態で使う関数）

OpenTP1 のライブラリ関数名	SUP		SPP			MHP		オフラインの業務をする UAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
dc_adm_get_nd_status_begin※	○	○	○	○	○	○	○	－
dc_adm_get_nd_status_next※	○	○	○	○	○	○	○	－
dc_adm_get_nd_status※	○	○	○	○	○	○	○	－
dc_adm_get_nd_status_done※	○	○	○	○	○	○	○	－
dc_adm_get_nodeconf_begin※	○	○	○	○	○	○	○	－
dc_adm_get_nodeconf_next※	○	○	○	○	○	○	○	－
dc_adm_get_nodeconf_done※	○	○	○	○	○	○	○	－
dc_adm_get_node_id※	○	○	○	○	○	○	○	－
dc_adm_get_sv_status_begin	○	○	○	○	○	○	○	－
dc_adm_get_sv_status_next	○	○	○	○	○	○	○	－
dc_adm_get_sv_status	○	○	○	○	○	○	○	－
dc_adm_get_sv_status_done	○	○	○	○	○	○	○	－
dc_uto_test_status	○	○	○	○	○	○	○	－

（凡例）

○：該当する条件で使えます。

－：該当する条件では使えません。

注※

この関数を使う UAP があるノードには，TP1/Multi が組み込まれていることが前提です。

1.5　アプリケーションプログラムのデバッグとテスタ

OpenTP1 では，作成した UAP を業務に使う前に，動作確認のテストができます。UAP をテストする機能を **UAP テスタ機能**といいます。

UAP テスタ機能を使うと，業務に使っている資源をテストのために変更する必要がなくなります。また，UAP テスタ機能はオペレータがコマンドを入力して確認する形式でテストできるため，テストしたい項目を重点的に確認できます。

1.5.1　UAP テスタ機能の種類

OpenTP1 の UAP テスタ機能には，使う目的別に，次のものがあります。

(1)　オフラインテスタ（TP1/Offline Tester）

オンライン業務用の UAP をオフライン環境でテストする機能です。OpenTP1 を稼働させていなくても，UAP の動作をテストできます。OpenTP1 の資源を使う前に，UAP を単体でテストする場合に使います。オフラインテスタでは，高級言語（C 言語，COBOL 言語）の**デバッガ**と連動できるので，テストとデバッグで UAP を二重にチェックできます。

オフラインテスタでは，SPP と MHP の動作をテストできます。

オフラインテスタを実行するマシンには，TP1/Offline Tester を組み込んであることが前提となります。

(2)　オンラインテスタ（TP1/Online Tester）

オンライン環境で UAP をテストする機能です。OpenTP1 のシステムサービスと連携して，UAP の動作をテストできます。OpenTP1 の UAP を統合してテストするときに使います。オンラインテスタでは，高級言語（C 言語，COBOL 言語）のデバッガと連動できるので，テストとデバッグで UAP を二重にチェックできます。

オンラインテスタでは，SUP と SPP の動作をテストできます。さらに，MHP を SPP として動作をテストできます。

オンラインテスタを実行するマシンには，TP1/Online Tester を組み込んであることが前提となります。

(3)　MCF オンラインテスタ（TP1/Message Control/Tester）

オンライン環境で UAP をテストする機能です。TP1/Message Control と連携して，MHP をテストするときに使います。OpenTP1 のシステムサービスと MCF のシステムサービスを使って，UAP をテストできます。

MCF オンラインテスタを実行するマシンには，TP1/Message Control/Tester を組み込んであることが前提となります。また，オンラインテスタの UAP トレース機能を使う場合は，TP1/Online Tester を組み込んであることが前提となります。

1.5.2 テストできるアプリケーションプログラム

UAP テスタ機能でテストできるのは，SUP，SPP，および MHP です。UAP テスタ機能によって，テストできる内容は異なります。

OpenTP1 クライアント機能（TP1/Client）で使う UAP（CUP）では，CUP からサービスを要求した SPP をテストモードで起動できます。

オフラインの業務をする UAP は，UAP テスタ機能でテストできません。

UAP テスタ機能については，マニュアル「OpenTP1 テスタ・UAP トレース使用の手引」を参照してください。

1.5.3 ユーザサーバのテスト状態の報告

OpenTP1 でオンラインテスタ（TP1/Online Tester）を使っている場合，ユーザサーバから状態を検知できます。テスト状態を検知するときは，dc_uto_test_status 関数【CBLDCUTO('T-STATUS')】を使います。dc_uto_test_status 関数でわかる項目を次に示します。

- テストユーザ ID（環境変数 DCUTOKEY に設定した値）
- ユーザサーバがテストモードで稼働しているかどうか
- グローバルトランザクションの処理状態
- ユーザサービス定義に指定した，次の項目
  test_mode オペランドに指定したテスト種別
  test_transaction_commit オペランドで指定したトランザクションの同期点の扱い
  test_adm_call_command オペランドで指定したコマンドの実行結果の扱い

2 OpenTP1 の基本機能（TP1/Server Base，TP1/LiNK）

TP1/Server Base，または TP1/LiNK を使っているノードで使うアプリケーションプログラムの機能について説明します。

この章では，各機能を **C 言語の関数名**で説明します。C 言語の関数名に対応する COBOL 言語の API は，関数を最初に説明する個所に【 】で囲んで表記します。それ以降は，C 言語の関数名に統一して説明します。COBOL 言語の API がない関数の場合は，【 】の表記はしません。

2.1　リモートプロシジャコール

OpenTP1 の UAP では，サービスを提供する UAP がネットワークのどのノードにあるかを意識しなくても，UAP のサービスを関数呼び出しのように要求できます。このようなプロセス間通信を**リモートプロシジャコール（RPC）**といいます。OpenTP1 の UAP で使えるリモートプロシジャコールには，次に示す 3 種類があります。

- OpenTP1 独自のインタフェース
- XATMI インタフェース（X/Open の仕様に準拠した RPC）
- TxRPC インタフェース（X/Open の仕様に準拠した RPC）

通信プロトコルに TCP/IP を使う場合には，上記 3 種類のリモートプロシジャコールを使えます。通信プロトコルに OSI TP を使う場合には，**XATMI インタフェースだけ**を使えます。OSI TP を使う場合のリモートプロシジャコールについては，「2.10　OSI TP を使ったクライアント／サーバ形態の通信」および「5.1　XATMI インタフェース（クライアント／サーバ形態の通信）」を参照してください。

ここでは，OpenTP1 独自のインタフェースの RPC について説明します。XATMI インタフェースについては「5.1　XATMI インタフェース（クライアント／サーバ形態の通信）」を，TxRPC インタフェースについては「6.1　TxRPC インタフェースの通信」を参照してください。

注意事項

システム共通定義の all_node オペランドで指定したドメイン以外の OpenTP1 システムにトランザクショナル RPC を行う場合，自ドメインおよび他ドメイン内のすべての OpenTP1 システムのノード識別子(システム共通定義の node_id オペランド)は一意にする必要があります。また，すべての OpenTP1 システムは，バージョン 03-02 以降にする必要があります。これらの条件を満たしていないと，トランザクションが正しく回復できなくなる場合があります。

2.1.1　リモートプロシジャコールの実現方法

クライアント UAP から，サービスを要求する関数を使って SPP のサービスを要求できます。UAP からサービスを要求するときは，サーバ UAP の**サービスグループ名**※と**サービス名**を引数に設定した **dc_rpc_call 関数【CBLDCRPC('CALL ')】**を呼び出します。要求されるサービスが，クライアント UAP と別のノードにあっても同じノードにあってもかまいません。要求されるサービスがどのノードにあるかは，OpenTP1 のネームサービスで管理しているので，UAP で意識する必要はありません。

注※

サービスグループ名をドメイン修飾して設定することで，設定したドメイン内のサーバ UAP へサービスを要求することもできます。ドメイン修飾したサービス要求については，「2.1.18　ドメイン修飾をしたサービス要求」を参照してください。

OpenTP1 では，サーバ UAP はサービス提供プログラム（**SPP**）です。SPP のサービスを要求できるクライアント UAP は，SUP，SPP，MHP，CUP です。

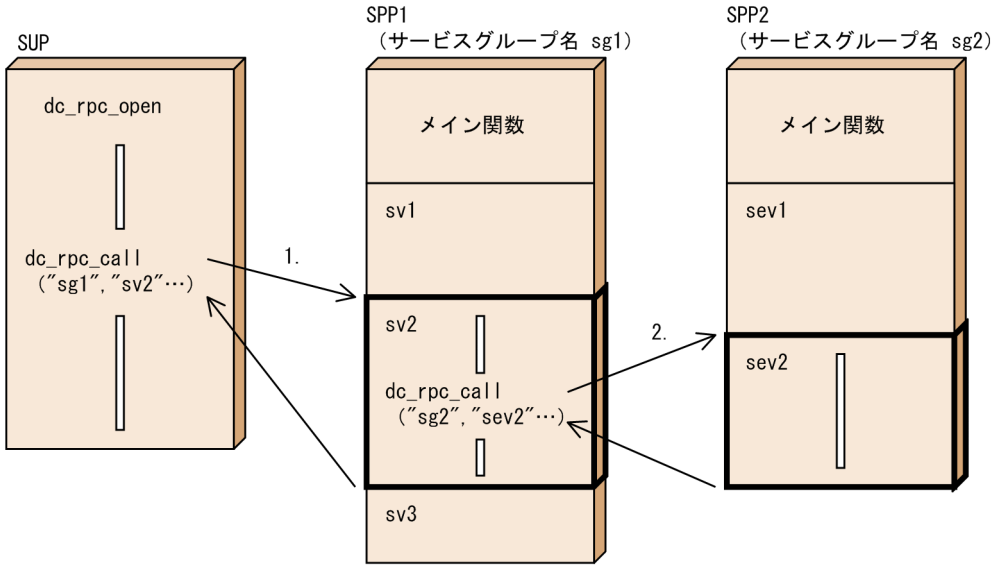
サーバ UAP は，**OpenTP1 の開始と一緒に開始（自動起動）**しておくか，**OpenTP1 の開始後に dcsvstart コマンドで開始（手動起動）**しておきます。開始しておくことで，サーバ UAP はサービスを提供できる状態になります。開始していないサーバ UAP にサービスを要求すると，dc_rpc_call 関数はエラーリターンします。

クライアント UAP は，開始したサーバ UAP のプロセスが稼働しているかどうかに関係なく，dc_rpc_call 関数でサービスを要求できます。サービスを要求したときに，指定したサーバ UAP のプロセスが稼働していなくても，OpenTP1 が自動的にプロセスを起動します。

MHP から RPC でサービスは要求できますが，MHP のサービス関数へはサービスを要求できません。また，オフラインの業務をする UAP では，RPC を使えません。

RPC を使った通信のクライアント／サーバの関係を次の図に示します。

図 2-1　RPC を使った通信のクライアント／サーバの関係



（凡例）
1. クライアントUAPから，サービスグループ名sg1 サービス名sv2 のサービスを要求します。
2. サービスを要求されたSPP1のサービスsv2から，サービスグループ名sg2 サービス名sev2のサービスを要求できます。

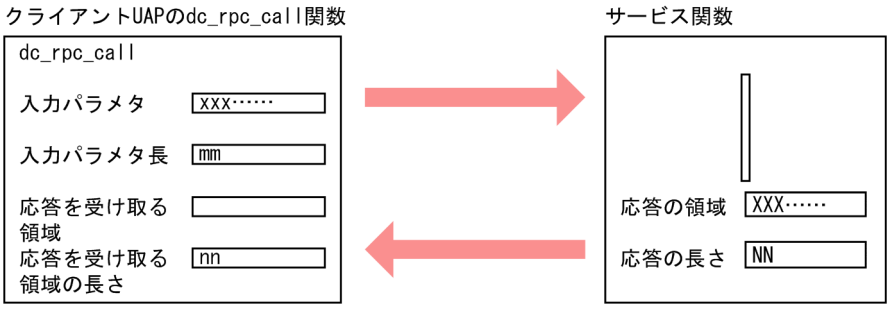
2.1.2　リモートプロシジャコールでのデータの受け渡し

クライアント UAP からサービスを要求するときには，dc_rpc_call 関数の引数に**入力パラメタ，入力パラメタ長，応答を格納する領域，応答を受け取る領域の長さ**を設定します。

サービス関数では，応答を格納する領域に**応答**を，応答を受け取る領域の長さに**応答の長さ**を設定してクライアント UAP に返します。

リモートプロシジャコールのデータの受け渡しを次の図に示します。

図 2-2　リモートプロシジャコールのデータの受け渡し

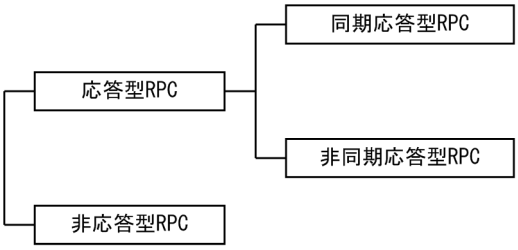


注　サービス関数から返す応答の長さ（NNの値）は，dc_rpc_call関数で設定した値（nnの値）と同じか，または小さい値となるようにしてください。

2.1.3　リモートプロシジャコールの形態

RPC には次の形態があります。RPC の形態は，クライアント UAP の dc_rpc_call 関数にフラグで設定します。RPC の形態を次の図に示します。

図 2-3　RPC の形態



- 応答型 RPC

  サーバ UAP の処理結果をクライアント UAP に返す RPC です。応答型 RPC には，dc_rpc_call 関数を呼び出してから，サーバ UAP の処理結果が返ってくるのを待つ**同期応答型 RPC** と，処理結果を非同期に受信する**非同期応答型 RPC** があります。

- 非応答型 RPC

  サーバ UAP の処理結果をクライアント UAP に返さない RPC です。dc_rpc_call 関数の呼び出し後にリターンして，処理を続けます。クライアント UAP ではサーバ UAP の処理結果を受信できません。

トランザクション処理で RPC を使った場合の，同期点と RPC との関係については，「2.3.4　リモートプロシジャコールの形態と同期点の関係」を参照してください。

(1)　同期応答型 RPC

dc_rpc_call 関数を呼び出してから，サーバ UAP の処理結果が返ってくるのを待つ形態です。同期応答型の RPC を使うときは，dc_rpc_call 関数の flags に DCNOFLAGS（または DCRPC_CHAINED）を設定します。

(a) 同期応答型 RPC の時間監視

dc_rpc_call 関数を呼び出してから，応答が返るまでの時間は，次に示す値で監視されています。

- TP1/Server Base の場合：
  ユーザサービス定義の watch_time オペランドに指定した値
- TP1/LiNK の場合：
  180 秒

サーバ UAP の処理に時間が掛かって，指定した監視時間を過ぎてしまった場合は，dc_rpc_call 関数はエラーリターンします。

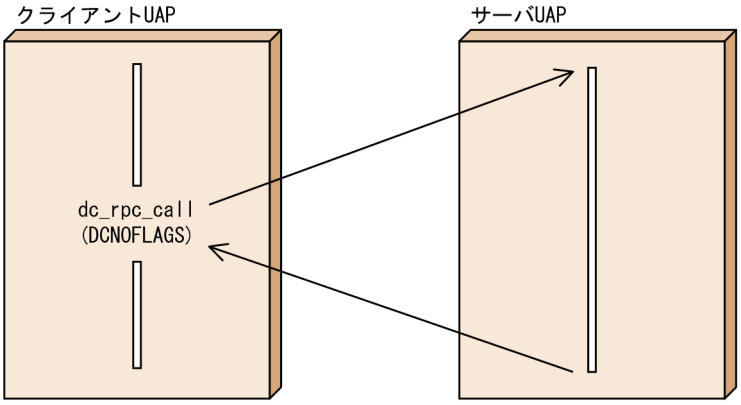
このときサーバ UAP では，クライアント UAP の応答待ち時間を認識しているため，クライアント UAP がタイムアウトしたあとにスケジューリングされたサービスは実行しないで廃棄し，実行中のサービスについては応答を返さずに処理を打ち切ります。サーバ UAP で，クライアント UAP のタイムアウトを検知したことによって，サービス要求を廃棄したことをメッセージ表示したい場合は，サーバ UAP のユーザサービス定義の rpc_extend_function オペランドに 00000008 を指定してください。

サーバ UAP が異常終了すると，dc_rpc_call 関数はすぐにエラーリターンします。ただし，次に示す場合には，指定した監視時間が過ぎてからエラーリターンします。

- サーバ UAP があるノードの OpenTP1 全体が異常終了した場合
- サービス要求のデータがサーバ UAP に届く前，またはサーバ UAP の処理が完了してからクライアント UAP に結果が届く前に障害が起こった場合

同期応答型 RPC を次の図に示します。

図 2-4　同期応答型 RPC



(2) 非同期応答型 RPC

dc_rpc_call 関数を呼び出してから，サーバ UAP の処理結果が返ってくるのを待たないで処理を続ける形態です。非同期応答型 RPC を複数回呼び出すことで，**RPC を並行処理**できます。

非同期応答型 RPC の **dc_rpc_call 関数**（flags に **DCRPC_NOWAIT** を設定）は，サービス要求後にリターンします。UAP ではリターン後に処理を続けます。

非同期応答型 RPC は，サービス要求がサーバに受け付けられたことを確認しないでリターンします。一つのクライアント UAP から同じサービスグループに複数の非同期応答型 RPC でサービスを要求した場合，サーバ側でこの要求順どおりにサービスを受け付けるとは限りません。

(a) 非同期応答型 RPC の応答の受信

サーバ UAP の処理結果は，**dc_rpc_poll_any_replies 関数【CBLDCRPC('POLLANYR')】**を呼び出して，非同期に受信します。処理結果は，dc_rpc_poll_any_replies 関数を使わないと受信できません。

非同期応答型 RPC の応答を受信するときは，どの応答を受信するか特定できます。特定する場合は，非同期応答型 RPC のサービス要求をした dc_rpc_call 関数がリターンしたときに返された正の整数（**記述子**）を，dc_rpc_poll_any_replies 関数の引数に設定します。この設定をすると，記述子に該当する非同期応答型 RPC の応答を受信します。

受信する応答を特定しない場合は，応答が戻ってきた順に受信します。応答を特定しない場合，dc_rpc_poll_any_replies 関数が正常に終了すると，受信した非同期応答の記述子と同じ値をリターンします。

非同期応答型 RPC の dc_rpc_call 関数を呼び出した数よりも多く dc_rpc_poll_any_replies 関数を呼び出した場合は，エラーリターンします。

サービス要求でエラーが起こった場合は，dc_rpc_poll_any_replies 関数にエラーリターンします。

flags に DCRPC_NOWAIT を設定した dc_rpc_call 関数を呼び出し，dc_rpc_call 関数に対する応答を受信する前にトランザクション同期点処理をする関数を呼び出すと，そのあとに呼び出す dc_rpc_poll_any_replies 関数は，DCRPCER_ALL_RECEIVED でエラーリターンし，応答を受信できません。非同期応答型 RPC の場合，トランザクション内で実行したかどうかに関係なく，該当するプロセスでのトランザクション同期点処理の実行前に応答を受信する必要があります。

(b) 非同期応答型 RPC の時間監視

非同期応答型 RPC の場合，ユーザサービス定義の watch_time に指定した値は参照されません。dc_rpc_poll_any_replies 関数を呼び出してから，応答が返るまでの最大応答待ち時間は，引数 timeout に設定します。

サーバ UAP の処理に時間が掛かって，設定した監視時間を過ぎてしまった場合は，dc_rpc_poll_any_replies 関数はエラーリターンします。

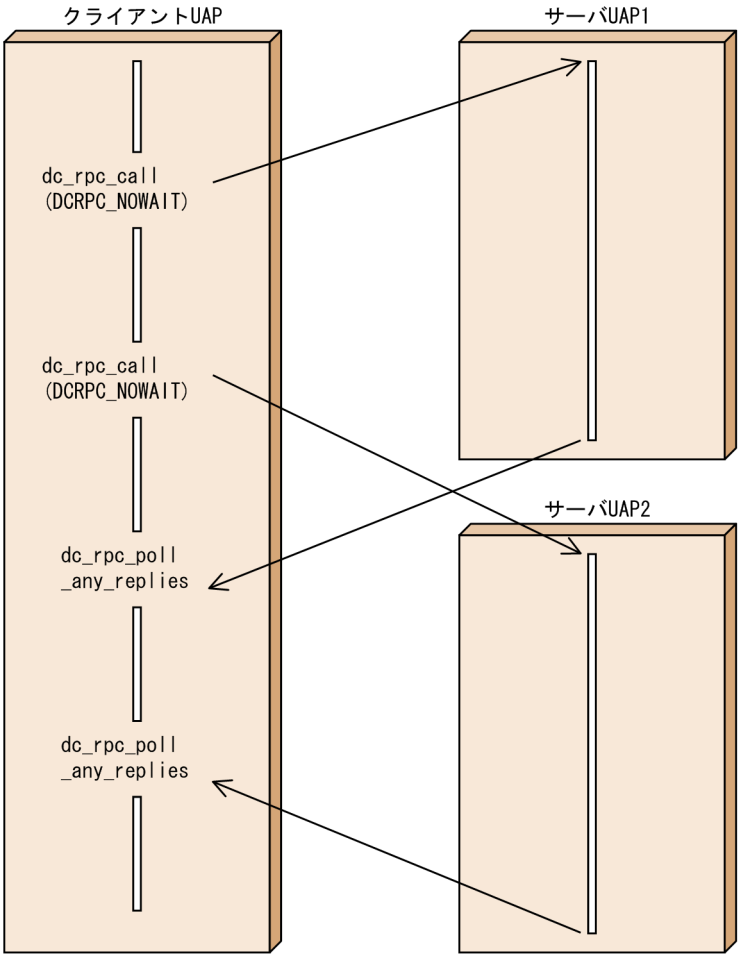
サーバ UAP が異常終了すると，dc_rpc_poll_any_replies 関数はすぐにエラーリターンします。ただし，次に示す場合には，設定した監視時間が過ぎてからエラーリターンします。

- サーバ UAP があるノードの OpenTP1 全体が異常終了した場合

- サービス要求のデータがサーバ UAP に届く前，またはサーバ UAP の処理が完了してからクライアント UAP に結果が届く前に障害が起こった場合

処理結果の非同期受信を次の図に示します。

図 2-5　非同期応答型 RPC（処理結果の非同期受信）



(c)　処理結果の受信を拒否

非同期応答型 RPC で，まだ返ってきていない応答をこれ以上受信しない場合は，処理結果の受信を拒否する関数（dc_rpc_discard_further_replies 関数【CBLDCRPC('DISCARDF')】）を呼び出します。この関数を呼び出したあとに返ってきた応答は，受信されないで捨てられます。非同期応答型 RPC の結果を受信しない場合は，必ず処理結果の受信を拒否する関数を呼び出してください。呼び出さないと，ほかの非同期応答型 RPC の dc_rpc_poll_any_replies 関数が，不要な応答を受信してしまうことがあります。

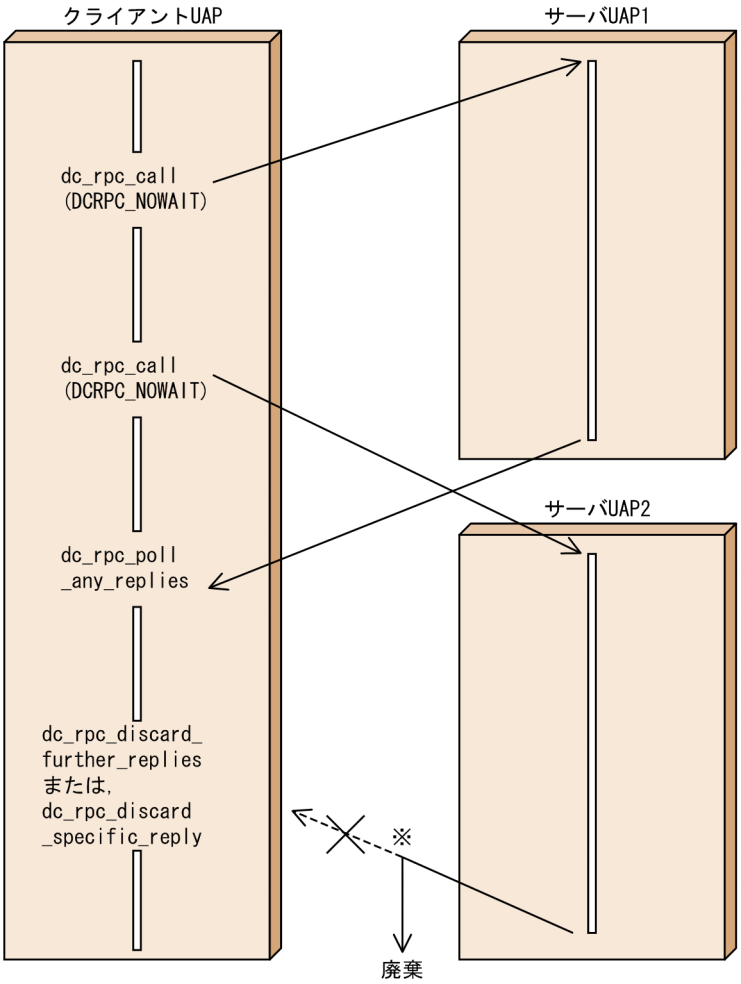
dc_rpc_discard_further_replies 関数を使うのは次のような場合です。

- 応答待ち時間切れになったあと，次の処理に移る前に，結果を保持しておくバッファをすぐに解放したい場合
- 非同期応答型 RPC を複数回呼び出して，そのうち最初の応答だけ必要な場合

また，非同期応答型 RPC で，まだ返ってきていない応答の中で特定の応答だけを受信しない場合は，特定の処理結果の受信を拒否する関数（dc_rpc_discard_specific_reply 関数【CBLDCRPC('DISCARDS')】）を呼び出します。この関数を呼び出したあとに返ってきた応答の中で，指定された記述子と同じ記述子を持つ応答は受信されないで捨てられます。

処理結果の受信の拒否を次の図に示します。

図 2-6　非同期応答型 RPC（処理結果の受信を拒否）



注※　dc_rpc_discard_further_replies関数を呼び出したあとに戻ってきた処理結果は，受信されないで捨てられます。
dc_rpc_discard_specific_reply関数を呼び出した場合，指定された記述子と同じ記述子を持つ応答が捨てられます。

(d)　同期点との関係

非同期応答型 RPC をトランザクション内で呼び出した場合，同期点処理したあとは非同期に応答を受信できません。同期点と非同期応答型 RPC については，「2.3.4　リモートプロシジャコールの形態と同期点の関係」を参照してください。

(3) 非応答型 RPC

サーバ UAP の処理結果をクライアント UAP に返さない RPC です。クライアント UAP ではサーバ UAP の処理結果を受信できません。

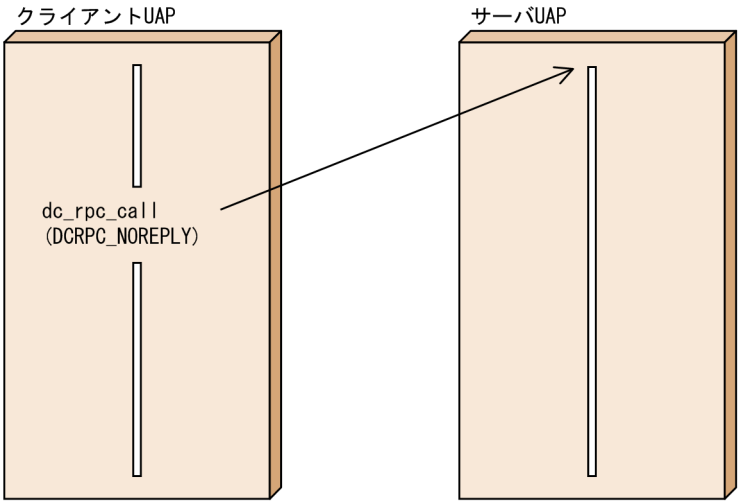
非応答型 RPC の**時間監視はしません**。

非応答型 RPC の **dc_rpc_call 関数**（flags に **DCRPC_NOREPLY** を設定）は，サービス要求後にリターンします。UAP ではリターン後は処理を続けます。サーバ UAP の処理結果は受信できません。

非応答型 RPC は，サービス要求がサーバに受け付けられたことを確認しないでリターンします。このことによって，通信障害などでサービス要求が消失しても，クライアント UAP で認識することはできません。また，一つのクライアント UAP から同じサービスグループに複数の非応答型 RPC でサービスを要求した場合，サーバ側でこの要求順どおりにサービスを受け付けるとは限りません。

非応答型 RPC を次の図に示します。

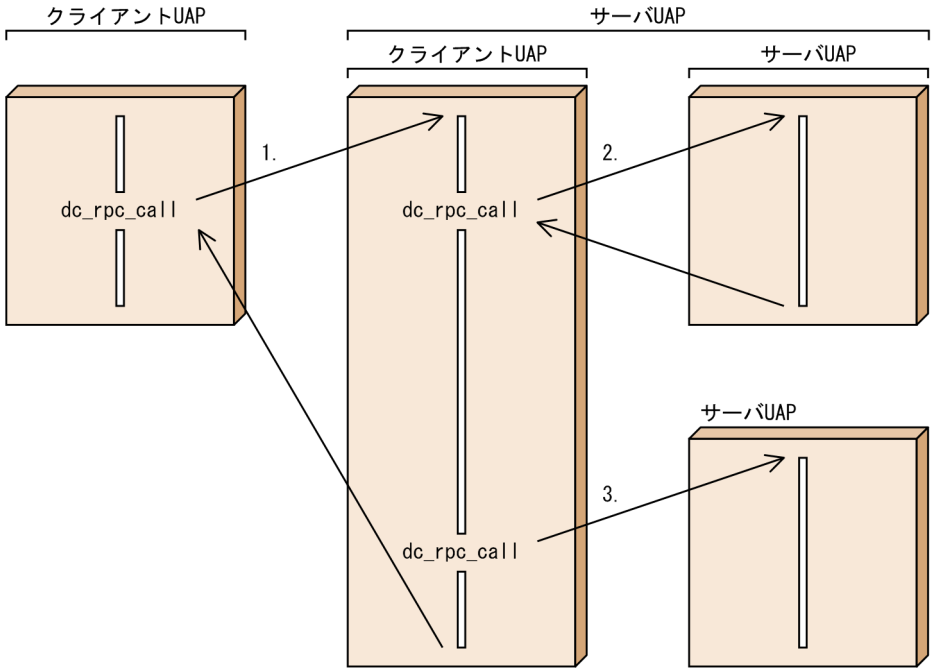
図 2-7　非応答型 RPC



2.1.4　サービスのネスト

クライアント UAP からサービス要求されたサーバ UAP から，さらにサーバ UAP を呼ぶことができます。このようなサーバ UAP のネストによって，サービス処理を分散化・階層化できます。RPC のネスト例を次の図に示します。

図 2-8　RPC のネスト例



（凡例）
1. 同期応答型RPC
2. 同期応答型RPCのネスト
3. 非応答型RPCのネスト

2.1.5　トランザクションの処理から非トランザクショナル RPC の発行

トランザクションの処理からサービスを要求した場合に，サービスを要求された UAP が**トランザクション属性**のときは，トランザクションの処理となります。このようなサービス要求を，トランザクション処理としないこともできます（**非トランザクショナル RPC**）。トランザクション処理としたくない場合は，dc_rpc_call 関数の引数にトランザクションでない RPC であることを指定します。

トランザクション属性については，「2.3.3　トランザクション属性の指定」を参照してください。

2.1.6　サービス要求のスケジュールプライオリティの設定

一つの処理から呼び出す複数のサービス要求に優先順位を付けたい場合，サービス要求ごとのプライオリティを設定できます。dc_rpc_call 関数を呼び出す直前に，

dc_rpc_set_service_prio 関数【CBLDCRPC('SETSVPRI')】を呼び出してサービス要求のプライオリティを設定します。これによって，サービス要求の優先度が，サーバ UAP 側のスケジュールキューを経由してサーバに通知されます。

dc_rpc_set_service_prio 関数を 1 度も使わない処理の場合は，スケジュールサービスの省略時解釈である 4 が，サービス要求のプライオリティとして指定されます。設定したスケジュールプライオリティは，**dc_rpc_get_service_prio 関数【CBLDCRPC('GETSVPRI')】**で参照できます。

キュー受信型サーバ（スケジュールサービスでスケジュールされる SPP）では，設定したサービス要求のプライオリティは，サーバ UAP のユーザサービス定義に，service_priority_control オペランドに Y（プライオリティを制御する）を指定している場合だけ有効です。サービスを要求する相手のサーバ UAP でプライオリティを制御していない場合は，この関数を呼び出しても無効になります。

ソケット受信型サーバ（スケジュールキューを経由しないでサービス要求を受信する SPP）では，クライアント UAP で設定したプライオリティに無条件に従います。

次に示すサービス要求に対して，dc_rpc_set_service_prio 関数を呼び出しても無効となります。

- 連鎖 RPC の 2 回目以降のサービス要求
- 連鎖 RPC を終了させるために呼び出す，同期応答型 RPC の dc_rpc_call 関数（flags に DCNOFLAGS を設定）

2.1.7 クライアント UAP のノードアドレスの取得

クライアント UAP へのサービスを制限したい場合，サーバ UAP でクライアント UAP を認識するため，クライアント UAP のプロセスが稼働するノードアドレスを取得できます。クライアント UAP のノードアドレスは，dc_rpc_get_callers_address 関数【CBLDCRPC('GETCLADR')】で取得します。

dc_rpc_get_callers_address 関数で返されたアドレスを使って，サービスの応答やエラーの応答などは送信できません。

dc_rpc_get_callers_address 関数は，サービス関数から呼び出してください。サービス関数以外から呼び出した場合の処理は保証しません。

2.1.8 サービス要求の応答待ち時間の参照と更新

UAP の処理の中で，サービス要求の応答待ち時間を一時的に変更できます。現在の応答待ち時間の設定を参照するときは dc_rpc_get_watch_time 関数【CBLDCRPC('GETWATCH')】を，変更するときは dc_rpc_set_watch_time 関数【CBLDCRPC('SETWATCH')】を使います。dc_rpc_set_watch_time 関数で変更した値は，該当の UAP で dc_rpc_close 関数を呼び出すまで有効です。

dc_rpc_get_watch_time 関数は，dc_rpc_set_watch_time 関数で変更したサービス応答待ち時間をリターンします。応答待ち時間を変更していない場合は，次に示す値がリターンされます。

- TP1/Server Base の場合：
  ユーザサービス定義の watch_time に指定した値
- TP1/LiNK の場合 ：
  180 秒

この関数で得られる値は，OpenTP1 の dc_rpc_call 関数の応答待ち時間として有効です。

サービス要求の応答待ち時間を，dc_rpc_set_watch_time 関数を呼び出す前の値に戻すときは，事前に呼び出している dc_rpc_get_watch_time 関数で返された元の値を，この関数で再設定してください。

dc_rpc_set_watch_time 関数は，関数を呼び出した UAP のサービス要求に影響するだけで，システム共通定義の watch_time オペランドに指定した値は変更しません。この関数で設定する値は，あとから呼び出す dc_rpc_call 関数にだけ影響します。

2.1.9　エラーが発生した非同期応答型 RPC 要求の記述子の取得

非同期応答を特定しない dc_rpc_poll_any_replies 関数【CBLDCRPC('POLLANYR')】がエラーリターンした直後に呼び出すことで，エラーが発生した非同期応答型 RPC 要求に対応する記述子を取得できます。

エラーが発生した非同期応答型 RPC 要求に対応する記述子は，dc_rpc_get_error_descriptor 関数【CBLDCRPC('GETERDES')】で取得します。

非同期応答の記述子を取得できるのは，SPP 側でエラーが発生した場合だけです。

dc_rpc_poll_any_replies 関数【CBLDCRPC('POLLANYR')】の呼び出し側でエラーが発生した場合には，dc_rpc_get_error_descriptor 関数【CBLDCRPC('GETERDES')】で非同期応答の記述子を取得できません。

2.1.10　CUP への一方通知

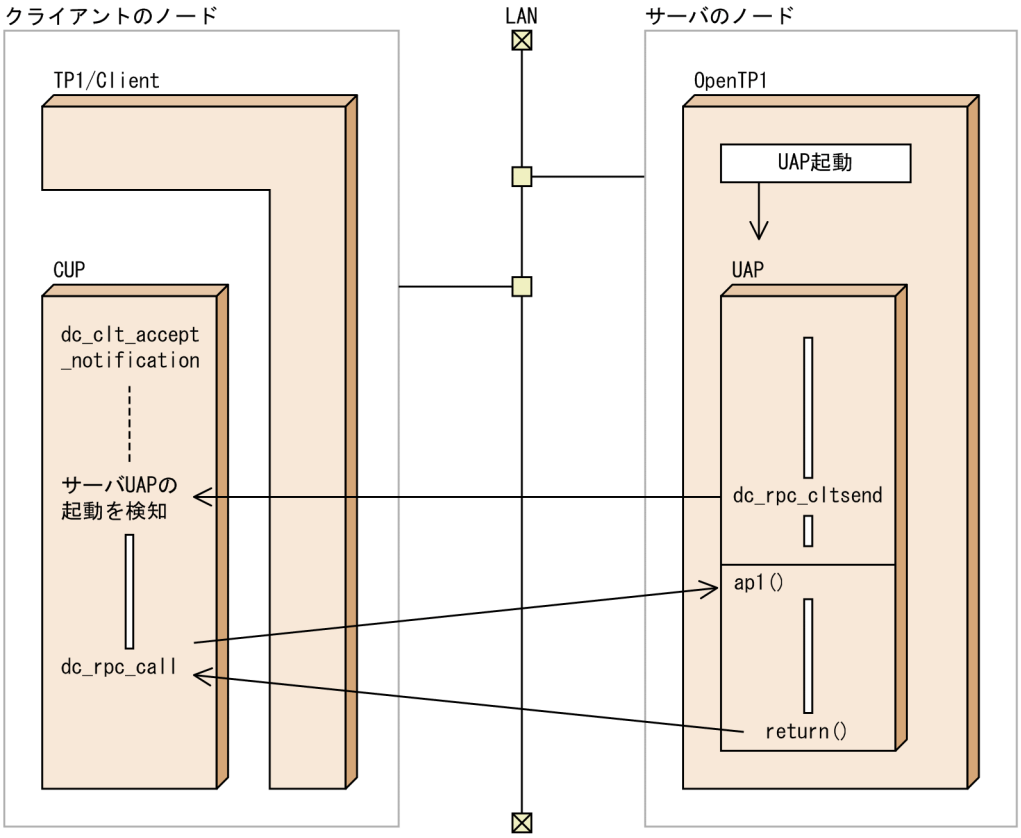
OpenTP1 のサーバ UAP から，TP1/Client のアプリケーションプログラム（CUP）へ UAP が開始したことを通知できます。CUP へは，dc_rpc_cltsend 関数【CBLDCRPC('CLTSEND ')】でデータを送って，サーバ UAP の開始を通知します。この機能を使って，サーバの起動完了を一斉にクライアントへ知らせることができます。

dc_rpc_cltsend 関数で通知したデータは，CUP の dc_clt_chained_accept_notification 関数または dc_clt_accept_notification 関数で受け取ります。CUP がデータを受け取ることで，TP1/Client はサーバが稼働中であることがわかります。その後，CUP からサーバへサービスを要求します。

dc_rpc_cltsend 関数は，通知する先の CUP が dc_clt_chained_accept_notification 関数または dc_clt_accept_notification 関数で通知を待っているときにだけ使えます。CUP が稼働していない場合には，dc_rpc_cltsend 関数はエラーリターンします。また，dc_rpc_cltsend 関数でデータを通知できるのは，CUP の dc_clt_chained_accept_notification 関数または dc_clt_accept_notification 関数だけです。そのほかのプロセス（サーバ UAP のプロセス）へは，データを送信できません。
dc_clt_chained_accept_notification 関数または dc_clt_accept_notification 関数については，マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」を参照してください。

CUP への一方通知の概要を次の図に示します。

図 2-9　CUP への一方通知の概要



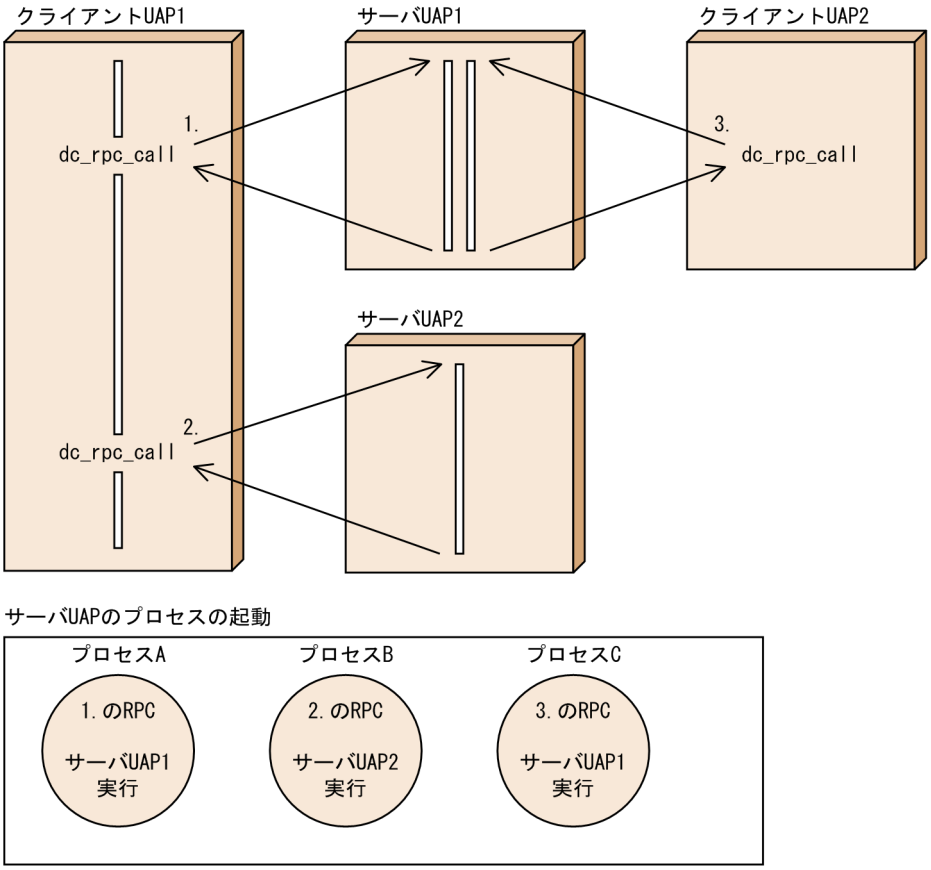
2.1.11　リモートプロシジャコールとサービスを実行するプロセスの関連

サービスを要求されたサーバ UAP は，クライアント UAP とは別のプロセスで実行されます。OpenTP1 では，一つのサーバ UAP を実行するためのプロセスを複数起動できる**マルチサーバ**を実現できます。マルチサーバの場合で RPC をネストさせると，ネストさせるサービス数だけプロセスが実行される場合があります。

同じサーバ UAP でも，クライアント UAP が異なれば，決まったプロセスで実行されるとは限りません。また，クライアント UAP と同じサービスグループに属するサービスを要求する場合でも，そのサービスグループを実行するための新しいプロセスが必要になります。マルチサーバを使う場合は，使用するプロセス数には余裕を持った値を指定しておいてください。

RPC とプロセスの関係を次の図に示します。

図 2-10　RPC とプロセスの関係



1. クライアント UAP1 からサーバ UAP1 にサービスを要求した場合，サーバ UAP1 はプロセス A で実行されます。
2. クライアント UAP1 からサーバ UAP2 にサービスを要求した場合，サーバ UAP2 はプロセス B で実行されます。
3. 新たなクライアント UAP2 からサーバ UAP1 にサービスを要求した場合，1.のサービス要求とは別の，プロセス C で実行されます。

(1)　連鎖 RPC

同じサービスグループに属するサービスを 2 回以上要求する同期応答型 RPC の場合に限り，そのサービスを以前と同じプロセスで実行できます。これを**連鎖 RPC** といいます。連鎖 RPC でサービスを要求すると，マルチサーバのサーバ UAP でも前回の RPC と同じプロセスで実行されるため，トランザクション処理に必要なプロセスを最小限にできます。UAP のプロセスはサービスグループごとに確保されるため，同じサービスグループに属していれば，異なるサービスに対しても連鎖 RPC でサービスを要求できます。

連鎖 RPC の処理は，トランザクションとしてもトランザクションとしなくても実行できます。トランザクションとして連鎖 RPC を実行する場合は，一つのグローバルトランザクションとして処理されます。

連鎖 RPC は，クライアント UAP のプロセス単位で保証されます。ただし，同じグローバルトランザクション内でも，クライアント UAP が異なれば，複数回呼び出されたサービスは同じプロセスで起動されることは保証されません。

(a) 連鎖 RPC の開始

連鎖 RPC となるサービス要求をする場合は，サービスを要求する dc_rpc_call 関数の flags に DCRPC_CHAINED を設定してください。この値を設定してサービスを要求したことで，サーバ UAP 側は連鎖 RPC であることを認識して，プロセスを確保します。2 回目以降のサービス要求の flags にも，DCRPC_CHAINED を設定します。

2 回目以降のサービス要求では，ユーザサーバおよびサービスの閉塞状態を検出できません。2 回目以降のサービス要求の実行中に異常が発生した場合は，ユーザサーバが閉塞します。このとき，サービス単位には閉塞できません。

(b) 連鎖 RPC の終了

連鎖 RPC は，次のどれかの方法で終了します。

- ユーザサービス定義の rpc_extend_function オペランドに 00000002 を指定していない場合は，連鎖 RPC を実行しているサービスグループに対して，同期応答型 RPC（flags に DCNOFLAGS を設定）でサービス要求する。
- トランザクション実行中に開始した連鎖 RPC の場合は，連鎖 RPC を実行しているグローバルトランザクションを同期点処理（コミット，またはロールバック）で完了させる。
- ユーザサービス定義の rpc_extend_function オペランドに 00000002 を指定している場合は，トランザクション実行中に開始した非トランザクションの連鎖 RPC が，同期点処理を実行したあと，連鎖 RPC を実行しているサービスグループに対して同期応答型 RPC（flags に DCNOFLAGS を設定）でサービス要求する。

(c) 連鎖 RPC の時間監視

連鎖 RPC の処理中に通信障害などで次のサービス要求が受け取れないと，SPP がプロセスを確保したままになってしまうことがあります。これを防ぐため，連鎖 RPC で実行している SPP では，応答を返してから次のサービス要求，または連鎖 RPC の終了要求が来るまでの時間（最大時間間隔）を次に示す値で監視しています。

- TP1/Server Base の場合：
  ユーザサービス定義の watch_next_chain_time オペランドに指定した値
- TP1/LiNK の場合：
  180 秒

上記の監視時間を過ぎても次のサービス要求，または連鎖 RPC の終了要求が来ない場合は，OpenTP1 はクライアント UAP で障害が起こったものと見なして，該当する SPP のプロセスを異常終了させます。

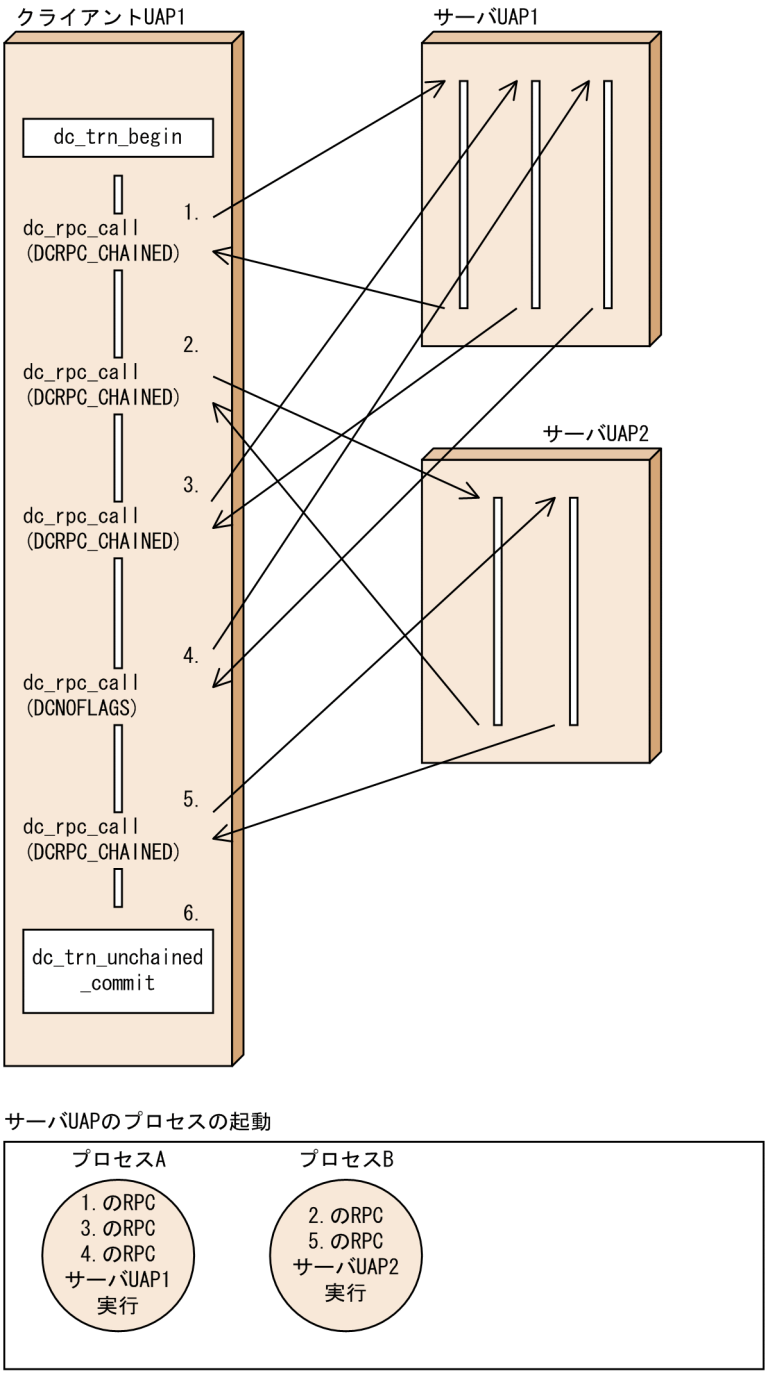
(d) ソケット受信型サーバへの連鎖 RPC について

ソケット受信型サーバ（スケジュールキューを経由しないでサービス要求を受信する SPP）は，マルチサーバとして稼働できません。また，非常駐プロセスとしても稼働できません。ソケット受信型サーバは，一つの常駐プロセスで稼働しています。

ソケット受信型サーバに対して連鎖 RPC でのサービス要求をすると，サーバ UAP は，該当のクライアント UAP だけからしかサービス要求を受け付けなくなります。ソケット受信型サーバへ連鎖 RPC でのサービス要求は，できるだけしないようにしてください。

連鎖 RPC とプロセスの関係を次の図に示します。

図 2-11　連鎖 RPC とプロセスの関係



1. クライアント UAP1 からサーバ UAP1 に連鎖 RPC（flags に DCRPC_CHAINED を設定）でサービスを要求した場合，サーバ UAP1 はプロセス A で実行されます。

2. クライアント UAP1 からサーバ UAP2 に連鎖 RPC でサービスを要求した場合，サーバ UAP2 はプロセス B で実行されます。
3. クライアント UAP1 からサーバ UAP1 に，再び連鎖 RPC でサービスを要求した場合，1.のサービス要求と同じプロセス A で実行されます。
4. クライアント UAP1 からサーバ UAP1 に，同期応答型 RPC（flags に DCNOFLAGS を設定）でサービスを要求した場合，サーバ UAP1 はプロセス A で実行されます。そしてクライアント UAP1 とサーバ UAP1 の連鎖 RPC は終了します。
5. クライアント UAP1 からサーバ UAP2 に，連鎖 RPC でサービスを要求した場合，サーバ UAP2 はプロセス B で実行されます。
6. 同期点を取得すると，クライアント UAP1 とサーバ UAP2 の連鎖 RPC は終了します。

2.1.12　リカーシブコールを使うときの注意

実行中のサーバ UAP から，自分と同じサービスグループ名とサービス名を指定してサービスを要求できます。これを**リカーシブコール（再帰呼び出し）**といいます。リカーシブコールの場合でも，その同じサービスを実行するために新しいプロセスが必要になります。そのため，リカーシブコールを使う場合，サービスを要求する指定をした時点で，実行できるプロセスがなくなる場合があります。このとき，RPC のタイムアウトになるか，待ち時間を指定していないときは永久に待ちになります。リカーシブコールを使うときは，余裕あるプロセス数を指定してください。なお，リカーシブコールを使えるのはキュー受信型サーバです。ソケット受信型サーバはリカーシブコールを使えません。

グローバルトランザクションの構成要素である一つのトランザクションブランチの中でも，リカーシブコールはできます。ただし，クライアント UAP と同じサービスグループに属していても，要求されたサービスは別プロセスで別のトランザクションブランチとして実行されます。

(1)　リカーシブコールとシステム定義との関連

ユーザサービス定義の **balance_count オペランド**に設定した値によっては，リカーシブコールをしてもプロセスが増えないで，タイムアウトになる場合があります。次の場合には，必ず balance_count オペランドの値に 0 を指定してください。

- 非常駐プロセスだけで構成されるユーザサーバでリカーシブコールをする場合（例えば，parallel_count=0,2 の場合）。
- 一つの常駐プロセスと，ほかの非常駐プロセスで構成されるサーバでリカーシブコールをする場合（例えば，parallel_count=1,2 の場合）。

ユーザサービス定義の最大プロセス数に 1 を指定した（parallel_count=1）のサービスグループに属するサービスでは，リカーシブコールは使えません。

2.1.13　サービス関数のリトライ

デッドロックなど，リトライすれば正常に実行できるトラブルが起こった場合に，クライアント UAP にエラーを返さないでサーバ UAP のプロセスをリトライできます。サービス関数をリトライして，クライアント UAP からのサービス要求をやり直す手間を省きたいときに使います。

サービス関数をリトライする場合は，サービス関数で dc_rpc_service_retry 関数【CBLDCRPC('SVRETRY')】を呼び出します。その後サービス関数をリターンすると，同じプロセスで同じサービス関数が再起動されます。

サービス関数をリトライした場合は，リトライする前のサービス関数が設定した内容（応答を格納する領域と応答の長さ）は無効になります。

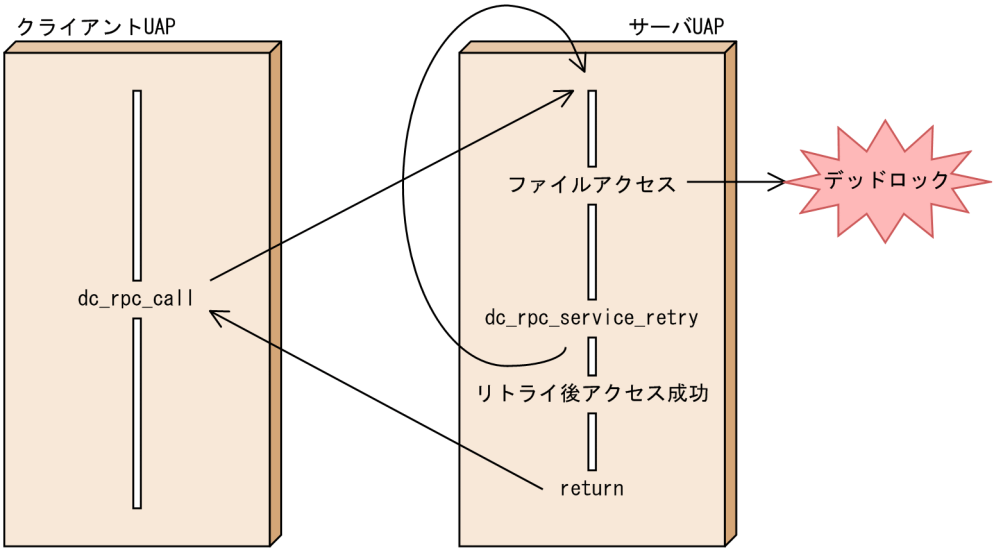
サービス関数をリトライする回数は，ユーザサービス定義の rpc_service_retry_count オペランドに指定します。このオペランドに指定した回数を超えた場合は，dc_rpc_service_retry 関数はエラーリターンします。このあとにリターンしたサービス関数はリトライされないで，応答の領域に設定した内容をクライアント UAP に返します。

dc_rpc_service_retry 関数を使う場合は，次に示す条件を満たしてください。この条件を満たさない場合は，関数がエラーリターンします。

- サービス関数の中で dc_rpc_service_retry 関数を呼び出していること。
- 実行中のサービス関数が，グローバルトランザクションの範囲でないこと。

サービス関数のリトライの概要を次の図に示します。

図 2-12　サービス関数のリトライの概要



2.1.14　ユーザデータの圧縮

ネットワーク上に送り出されるパケット数を削減し，ネットワークの負荷を軽減するため，RPC でやり取りするユーザデータを圧縮できます。ユーザデータを圧縮する場合は，クライアント側の OpenTP1 のシステム共通定義の rpc_datacomp オペランドに Y を指定します。

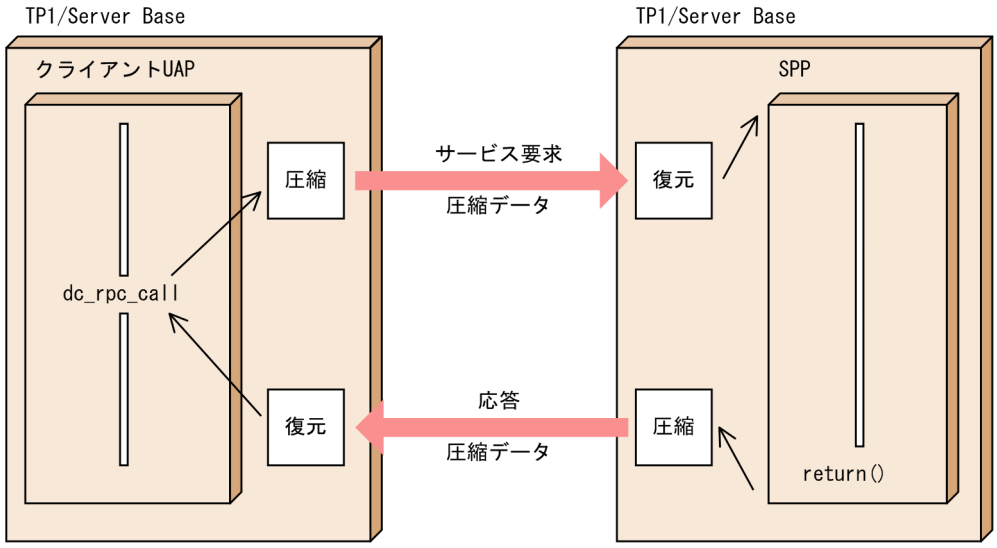
(1)　データ圧縮機能

データ圧縮機能を使用すると，クライアント側の OpenTP1 はクライアント UAP のサービス要求を圧縮してネットワーク上に送り出します。その要求に対し，SPP から返される応答もサーバ側の OpenTP1 で圧縮されてネットワーク上に送り出されます。その応答を受け取ったクライアント側の OpenTP1 は，圧縮データを復元してクライアント UAP に渡します。

rpc_datacomp オペランドの指定は，dc_rpc_call でサービスを要求するクライアント側で有効になります。つまり，クライアント側の OpenTP1 で rpc_datacomp オペランドに Y を指定していれば，サーバ側の OpenTP1 に rpc_datacomp オペランドに Y が指定されていなくても，サービス要求メッセージも応答メッセージもユーザデータを圧縮してネットワークに送り出します。反対に，クライアント側の OpenTP1 に rpc_datacomp オペランドに Y を指定していなければ，サーバ側の OpenTP1 に rpc_datacomp オペランドに Y が指定されていても，サービス要求メッセージ・応答メッセージともにユーザデータを圧縮しません。ただし，サーバ側の OpenTP1 がユーザデータの圧縮機能をサポートしている場合に限ります。

データ圧縮機能の概要を次の図に示します。

図 2-13　データ圧縮機能の概要



(2)　データ圧縮機能の効果

データ圧縮機能の効果は，ユーザデータの内容に依存します。ユーザデータ中に連続した同一文字が多く現れる場合は効果がありますが，同一文字が連続することのないユーザデータは圧縮の効果がありません。

クライアント側の OpenTP1 で rpc_datacomp オペランドに Y を指定していても，ユーザデータに圧縮効果がないときは，圧縮しないでサービス要求を送信します。しかし，応答メッセージが圧縮効果のある場合は，応答のユーザデータは圧縮して返送します。なお，サービス要求を圧縮しない場合でも，応答を圧縮して返すのは，クライアント側・サーバ側ともに TP1/Server Base のバージョンが 03-06 以降のときだけです。これ以外のバージョンでは，サービス要求を圧縮しない場合，応答は圧縮しないで返ります。

データ圧縮機能は，データ圧縮／復元のためのオーバヘッドがかかるため，事前にデータ圧縮による効果と性能への影響を評価してから使用してください。

2.1.15　サービス関数実行時間の監視

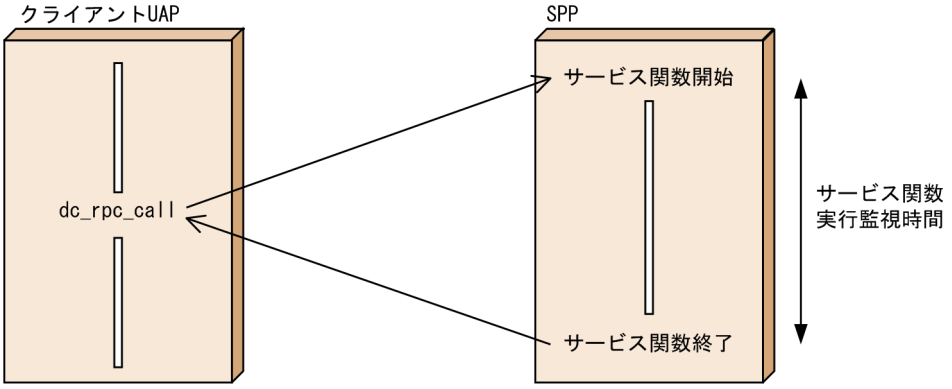
SPP のサービス関数開始から終了までの実行時間を監視できます。この機能はユーザ作成のサービス関数が，不具合によってダイナミックループしているような場合に，このサービスを打ち切るのに有効です。指定した監視時間を過ぎてもサービス関数がリターンしない場合，この SPP プロセスを強制終了させます。

サービス関数の実行時間を監視するにはユーザサービス定義の service_expiration_time オペランドに値を指定してください。

この機能は，SPP だけで動作し，MHP では動作しません。また，OSI TP プロトコルを使用した XATMI インタフェースや，DCE プロトコルを使用した TxRPC インタフェースで稼働する SPP では，この機能は動作しません。

サービス関数実行時間の監視の概要を次の図に示します。

図 2-14　サービス関数実行時間の監視の概要



2.1.16　マルチスケジューラ機能を使用した RPC

クライアント UAP が，他ノードのキュー受信型サーバ（スケジュールキューを使う SPP）に，マルチスケジューラ機能を使用してサービス要求することによって，一つの OpenTP1 システムで，次の 3 方式の RPC を混在して使用できます。

1. 通常の RPC

サービス要求先サーバが存在する OpenTP1 システムの中から一つをランダムに選択し，その OpenTP1 システムのマスタスケジューラデーモンにサービス要求を送信する方式。

2. マルチプルポート指定 RPC

   サービス要求先サーバが存在するすべての OpenTP1 システムのマルチスケジューラデーモンの中から一つをランダムに選択し，サービス要求を送信する方式。

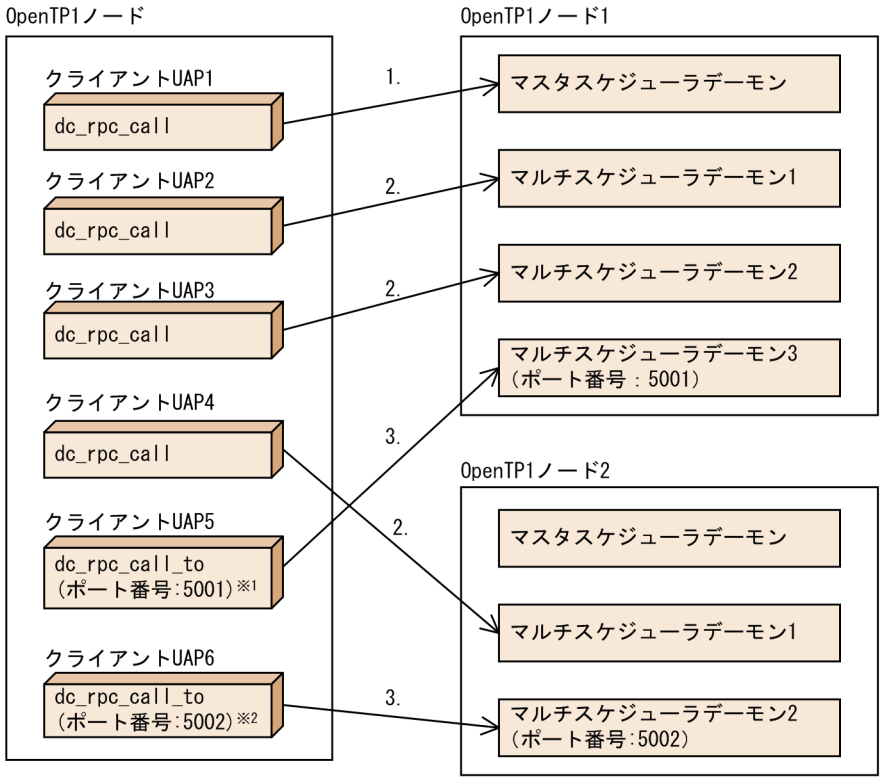
3. 通信先を指定した RPC

   dc_rpc_call_to 関数の引数に指定したポート番号のマルチスケジューラデーモンにサービス要求を送信する方式。

マルチスケジューラ機能の詳細については，「1.3.6(9)　マルチスケジューラ機能」を参照してください。なお，この機能は，TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証できませんので，ご了承ください。

マルチスケジューラ機能を使用した RPC の概要を次の図に示します。

図 2-15　マルチスケジューラ機能を使用した RPC の概要



（凡例）
1. ：通常のRPC（マルチスケジューラ機能を使用していない）
2. ：マルチプルポート指定RPC
3. ：通信先を指定したRPC

注※1　dc_rpc_call_to関数の引数に，OpenTP1ノード1にあるマルチスケジューラデーモン3のポート番号（5001）を指定。
注※2　dc_rpc_call_to関数の引数に，OpenTP1ノード2にあるマルチスケジューラデーモン2のポート番号（5002）を指定。

マルチスケジューラ機能を使用したサービス要求は，マルチスケジューラデーモンを起動しているノードにだけスケジュールします。ただし，サービス要求時に，指定したマルチスケジューラデーモンを起動し

ている OpenTP1 システムが存在しない場合は，マスタスケジューラデーモンにサービス要求を送信します。

ポート番号を指定してサービス要求する際に，マルチスケジューラデーモンのポート番号を指定した場合，指定したマルチスケジューラデーモン経由でサービス要求をスケジューリングします。

マルチスケジューラデーモンがサービス要求を受信した際，サービス要求先ユーザサーバが閉塞していた場合や終了中であった場合は，マルチスケジューラ機能を使用するほかのノードのマルチスケジューラデーモンにサービス要求を送信します。ほかのノードに該当するマルチスケジューラデーモンが存在しない場合は，マスタスケジューラデーモンにサービス要求を送信します。

オンライン中にマルチスケジューラデーモンが異常終了した場合は，OpenTP1 システムをダウンさせないで，異常終了したマルチスケジューラデーモンに該当するポート番号を割り当てて再起動します。ただし，再起動が 2 回失敗した場合は，OpenTP1 システムをダウンさせます。

2.1.17　通信先を指定した RPC

dc_rpc_call 関数を使ってサービスを要求する場合，要求するサービスがどこにあるかは，OpenTP1 のネームサービスで管理しているので，クライアント UAP で意識する必要はありません。

これに対し，**dc_rpc_call_to 関数**を使えば，特定のサービス要求先にサービスを要求できます。dc_rpc_call_to 関数では，ドメイン修飾をしてサービスを要求できません。それ以外は，dc_rpc_call 関数の機能と変わりません。

なお，この関数は，TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証できませんので，ご了承ください。この関数は，TP1/Server 管理下の C 言語で作成した UAP でだけ呼び出せます。COBOL 言語で作成した UAP では使用できません。

サービス要求先を特定するには，dc_rpc_call_to 関数の引数に次のどれかを指定する必要があります。

1. **ホスト名指定**

   /etc/hosts ファイル，または DNS などで IP アドレスとマッピングできるホスト名を指定して，サービス要求先ノードを特定します。

   このとき，サービス要求先のシステム共通定義の name_port オペランドに指定した値と，サービス要求元（dc_rpc_call_to 関数を呼び出した側）の name_port オペランドに指定した値が同じであることが前提です。

2. **ノード識別子指定**

   システム共通定義の node_id オペランドに指定されているノード識別子を指定して，サービス要求先の OpenTP1 ノードを特定します。

   指定したノード識別子に対応するサービス要求先の OpenTP1 ノードのホスト名がグローバルドメイン※内にあることが前提です。

3. **ホスト名とポート番号の指定**

次の値を指定することでサービス要求先を特定します。

- /etc/hosts ファイル，または DNS などで IP アドレスとマッピングできるホスト名
- 上記で指定したホストにある OpenTP1 システムの，システム共通定義の name_port オペランドに指定したネームサービスのポート番号

このとき，サービス要求先の name_port オペランドに指定した値と，サービス要求元の name_port オペランドに指定した値は同じでなくてもまいません。

注※

ここでのグローバルドメインとは，次のノード名の集合を指します。

システム共通定義の name_domain_file_use オペランドに N を指定している場合

システム共通定義の all_node オペランド，all_node_ex オペランドで指定したノード名の集合です。

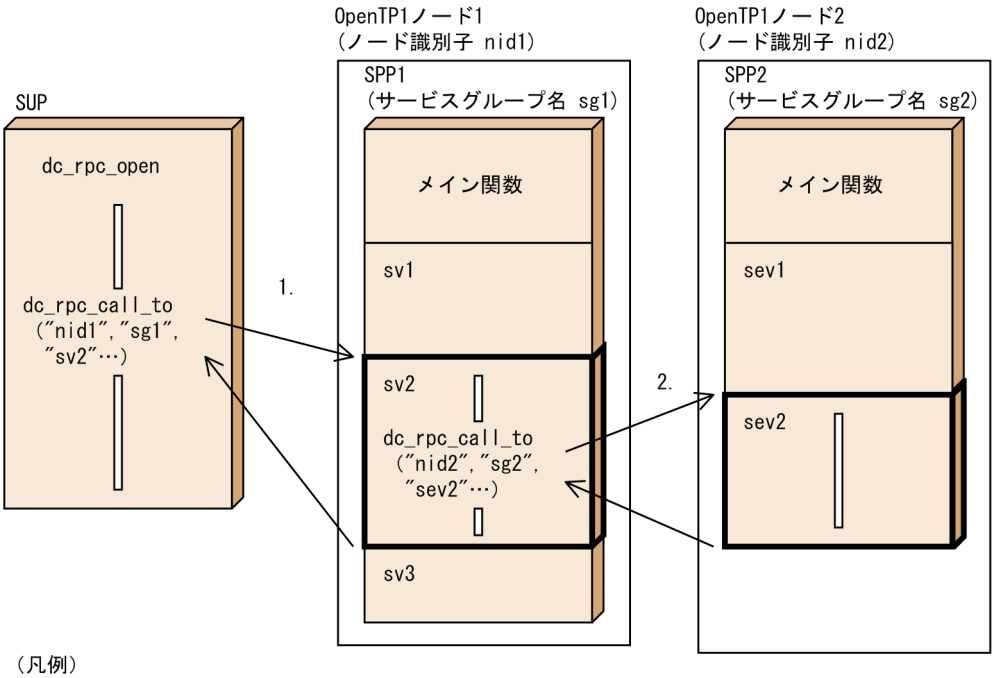
システム共通定義の name_domain_file_use オペランドに Y を指定している場合

ドメイン定義ファイルに指定したノード名の集合です。なお，ドメイン定義ファイルの格納場所は次のとおりです。

- all_node のドメイン定義ファイル
  $DCCONFPATH/dcnamndディレクトリ下
- all_node_ex のドメイン定義ファイル
  $DCCONFPATH/dcnamndexディレクトリ下

ノード識別子を指定して，サービス要求先を特定した dc_rpc_call_to 関数を使った通信の例を次の図に示します。

図 2-16　dc_rpc_call_to 関数を使った通信の例



（凡例）
1. クライアントUAPから，ノード識別子nid1　サービスグループ名sg1 サービス名sv2 のサービスを要求します。
2. サービスを要求されたSPP1のサービスsv2から，ノード識別子nid2　サービスグループ名sg2 サービス名sev2のサービスを要求できます。

2.1.18　ドメイン修飾をしたサービス要求

サービスを要求すると，OpenTP1 はシステムを構成するネットワークすべてを対象にして，通信相手を探します。そのため，ネットワークが大規模になると，サービス要求のスケジューリングに時間が掛かってしまいます。これを解決するため，ネットワークを DNS のドメインごとに区切ってサービスを要求できます。ドメインごとに区切ってサービスを要求すると，そのドメイン内で通信相手を探すため，スケジュールの性能が上がります。

ドメイン修飾をするときには，**DNS のドメイン名**を使います。このドメイン名を **dc_rpc_call 関数**の引数の**サービスグループ名**に続けて設定します。ドメイン修飾をしてサービスを要求する方法については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編にある dc_rpc_call 関数の説明を参照してください。

(1)　ドメイン修飾をするサービス要求の前提条件

ドメイン修飾をする場合の前提条件を次に示します。

1. ドメイン代表スケジュールサービスを起動するホスト名を，namdomainsetup コマンドで，DNS へ登録してあること。
2. ドメイン代表スケジュールサービスを起動する OpenTP1 のスケジュールサービス定義の scd_port オペランドに，スケジュールサービスのポート番号を指定してあること。

3. ドメイン修飾をしてサービスを要求する OpenTP1 を起動するホストの/etc/services に，2.で指定したスケジュールサービスのポート番号が登録してあること。

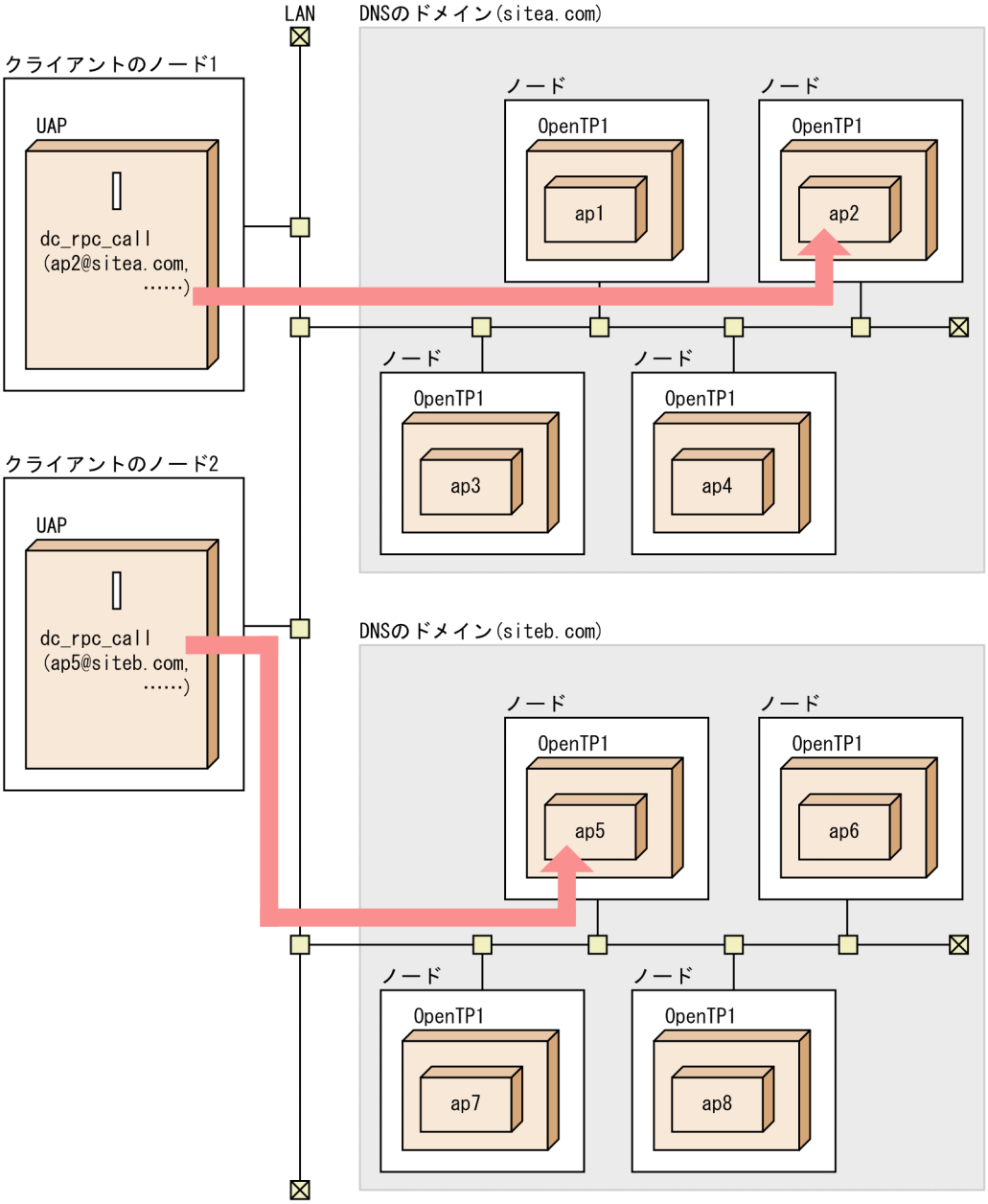
(2) ドメイン修飾をするサービス要求の制限

ドメイン修飾をしたサービス要求では，通常のサービス要求に比べて，次の制限があります。

1. キュー受信型サーバのサービスだけ，要求できます。ソケット受信型サーバのサービスは，ドメイン修飾をして要求できません。
2. トランザクション処理からサービスを要求しても，要求されたサービスの処理はトランザクションブランチにはなりません。

ドメイン修飾をしたサービス要求の概要を次の図に示します。

図 2-17　ドメイン修飾をしたサービス要求の概要



2.1.19　サービス関数とスタブの関係

SPP または MHP でサービス関数を作成する方法は，次の二つです。

1. スタブを使用して，該当するサービス関数を作成する方法
2. サービス関数動的ローディング機能を使用して，サービス関数を作成する方法

ここでは，上記の二つの方法について説明します。

(1) スタブを使用する場合

RPC を使って UAP 間で通信するときには，**スタブ**が必要です。スタブとは，クライアント UAP が指定した「サービスグループ名＋サービス名」とサーバ UAP のサービスとを対応づけるプログラムです。

スタブでは UAP の各サービスの入り口点（エントリポイント）を指定します。

エントリポイント名とは，C 言語の関数名であり，COBOL 言語のプログラム名または入り口名のことです。

サービス名とエントリポイント名は，1 対 1 で対応させてください。複数のサービス名に 1 つのエントリポイント名を対応させることはできません。

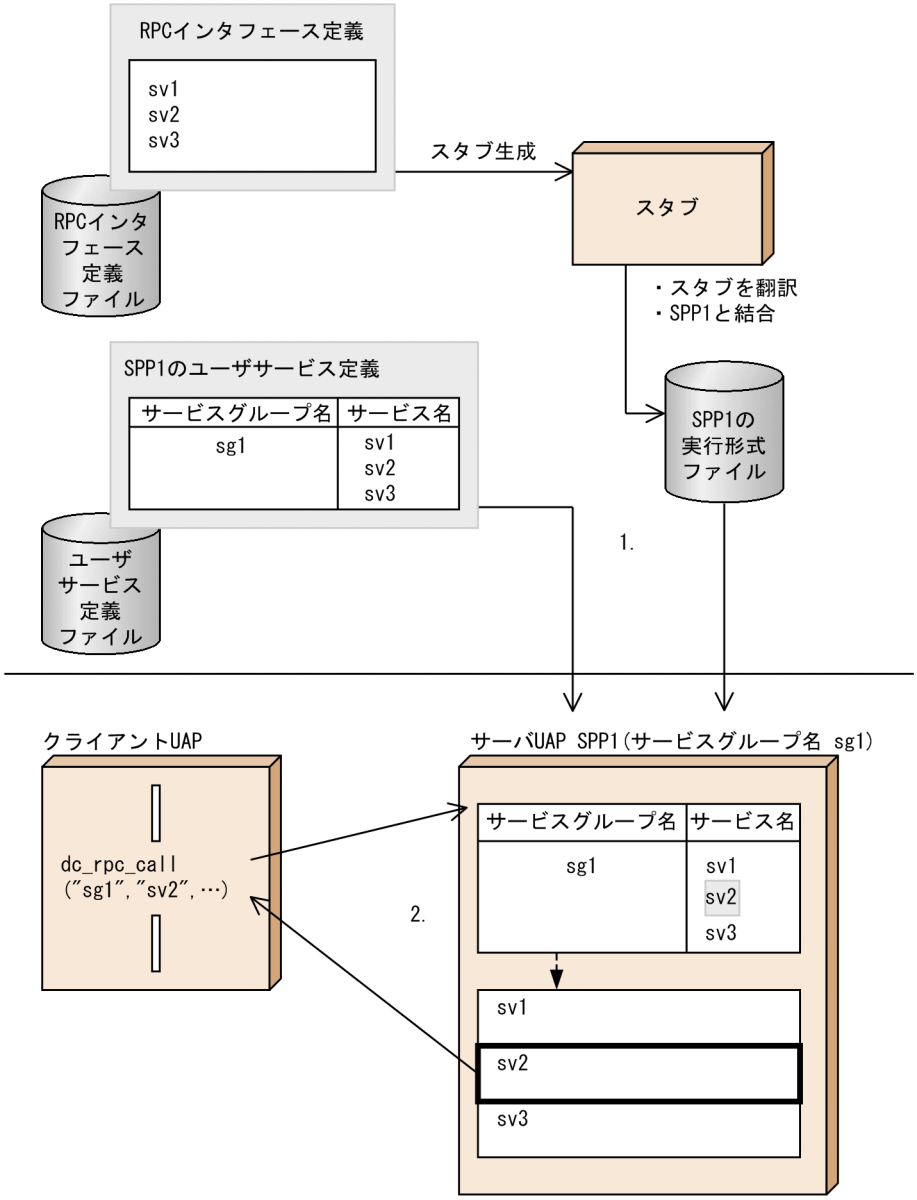
サービス名とエントリポイント名は，ユーザサービス定義で指定している service オペランドの名称と合わせてください。

スタブはサーバ UAP の作成時に，サーバ UAP のオブジェクトファイルと結合させます。

クライアント専用の UAP である SUP と，オフラインの業務をする UAP には，スタブを定義して結合させる必要はありません。

スタブを使用してサービス関数を作成する方法について，SPP の場合と MHP の場合に分けてそれぞれ以降の図に示します。

図 2-18　スタブを使用する場合（SPP）

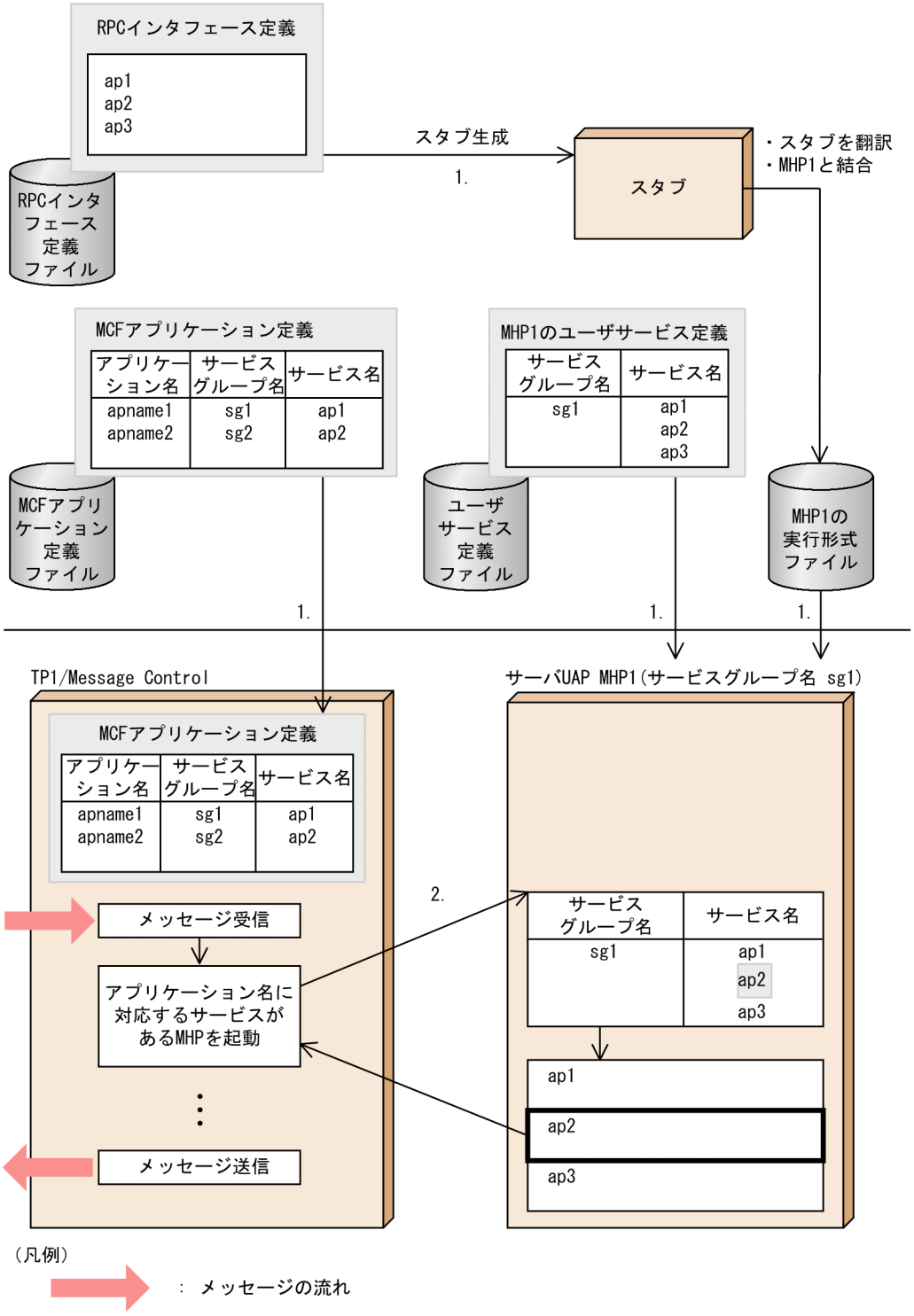


1. RPC インタフェース定義にサービス関数のエントリポイントを指定して，スタブを生成するコマンドでスタブを生成します。

   ユーザサービス定義では，サービスグループ名とサービス名を指定します。

2. 実行時には，サービスを要求されたサーバ UAP の実行形式ファイルでは，スタブとユーザサービス定義によって作成されたライブラリ部で，該当のサービスを検索します。その後サービスの処理をしてクライアント UAP に結果を返します。

図 2-19　スタブを使用する場合（MHP）



1. RPC インタフェース定義にサービス関数のエントリポイントを指定して，スタブを生成するコマンドでスタブを生成します。

   MCF アプリケーション定義では，アプリケーション名，サービスグループ名，およびサービス名を対応づけます。ユーザサービス定義では，サービスグループ名とサービス名を指定します。

2. 実行時には，TP1/Message Control の処理によって，MCF アプリケーション定義に基づきアプリケーション名に対応するサービス名が検索され，該当するサーバ UAP を起動します。サービスを要求され

たサーバ UAP の実行形式ファイルでは，スタブとユーザサービス定義によって作成されたライブラリ部で，該当のサービスを検索します。その後サービスの処理をし，処理の完了を TP1/Message Control に通知します。

(2) サービス関数動的ローディング機能を使用する場合

サービス関数動的ローディング機能を使う場合，UAP の各サービスの入り口点（エントリポイント）を指定した UAP ライブラリからサービス関数を取得するため，**スタブ**は不要です。その代わりに，サービス関数を共用ライブラリ化して UAP 共用ライブラリ※を作成する必要があります。これによって，UAP 共用ライブラリからサービス関数を取得できるとともに，複数のサービスをメイン関数にまとめる作業は不要になります。

注※

UAP 共用ライブラリとは，UAP のソースファイルを翻訳（コンパイル）して作成した UAP オブジェクトファイルを結合（リンケージ）して，共用ライブラリとしてまとめたものです。

サービス関数動的ローディング機能を使用してサービス関数を作成する方法について，SPP の場合と MHP の場合に分けてそれぞれ以降の図に示します。

図 2-20　サービス関数動的ローディング機能だけを使用する場合（SPP）

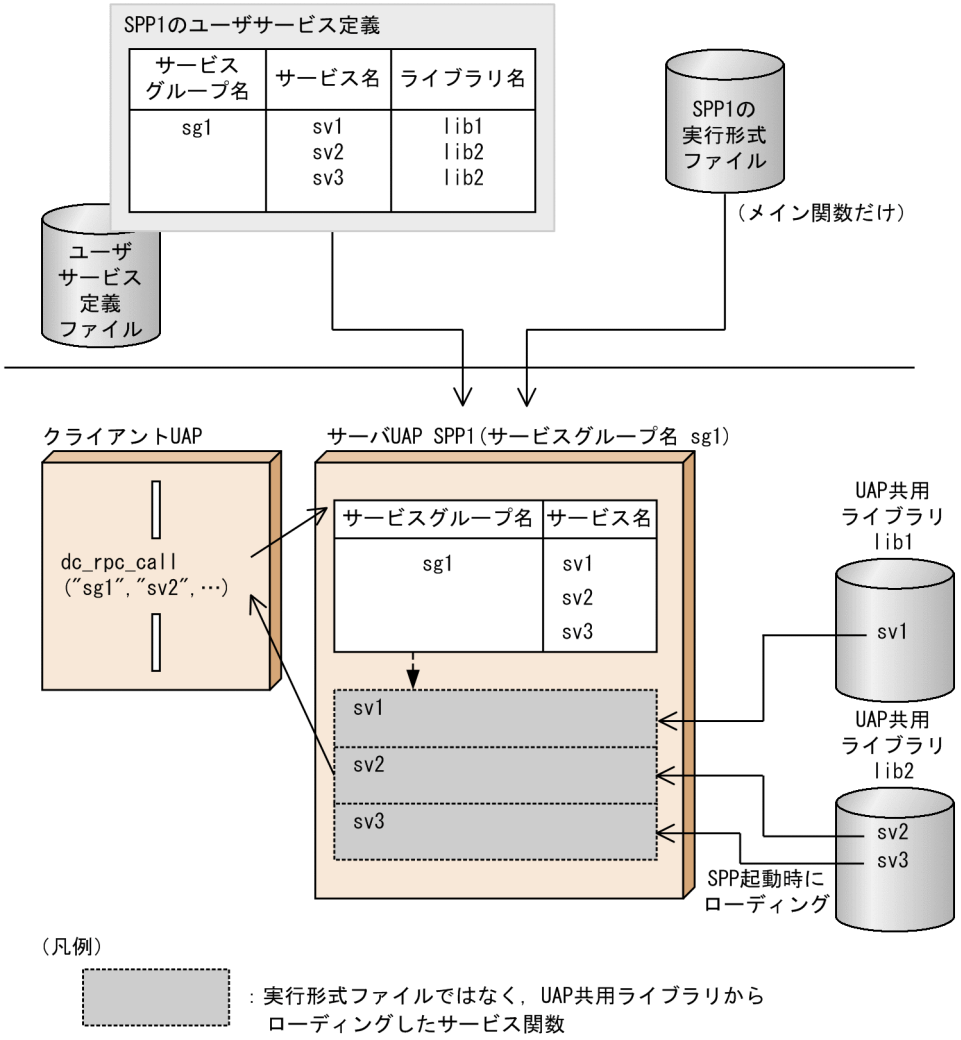
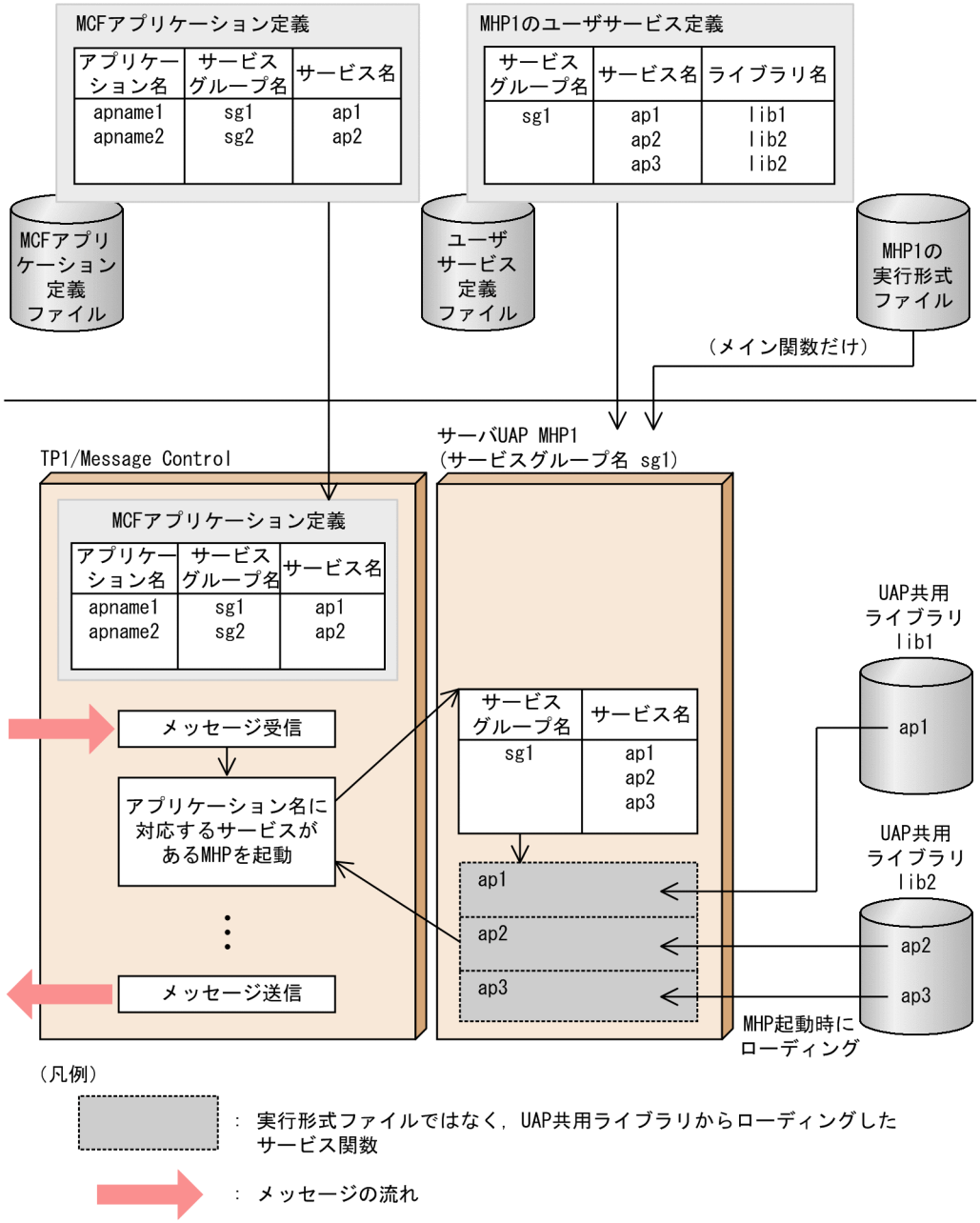


図 2-21　サービス関数動的ローディング機能だけを使用する場合（MHP）



なお，サービス関数動的ローディング機能は，スタブを使った UAP でも使用できます。この場合，スタブを使った UAP を変更しないで，サービス関数を追加できます。

サービス関数動的ローディング機能とスタブを併用してサービス関数を作成する方法について，SPP の場合と MHP の場合に分けてそれぞれ以降の図に示します。

図 2-22　サービス関数動的ローディング機能とスタブを併用する場合（SPP）

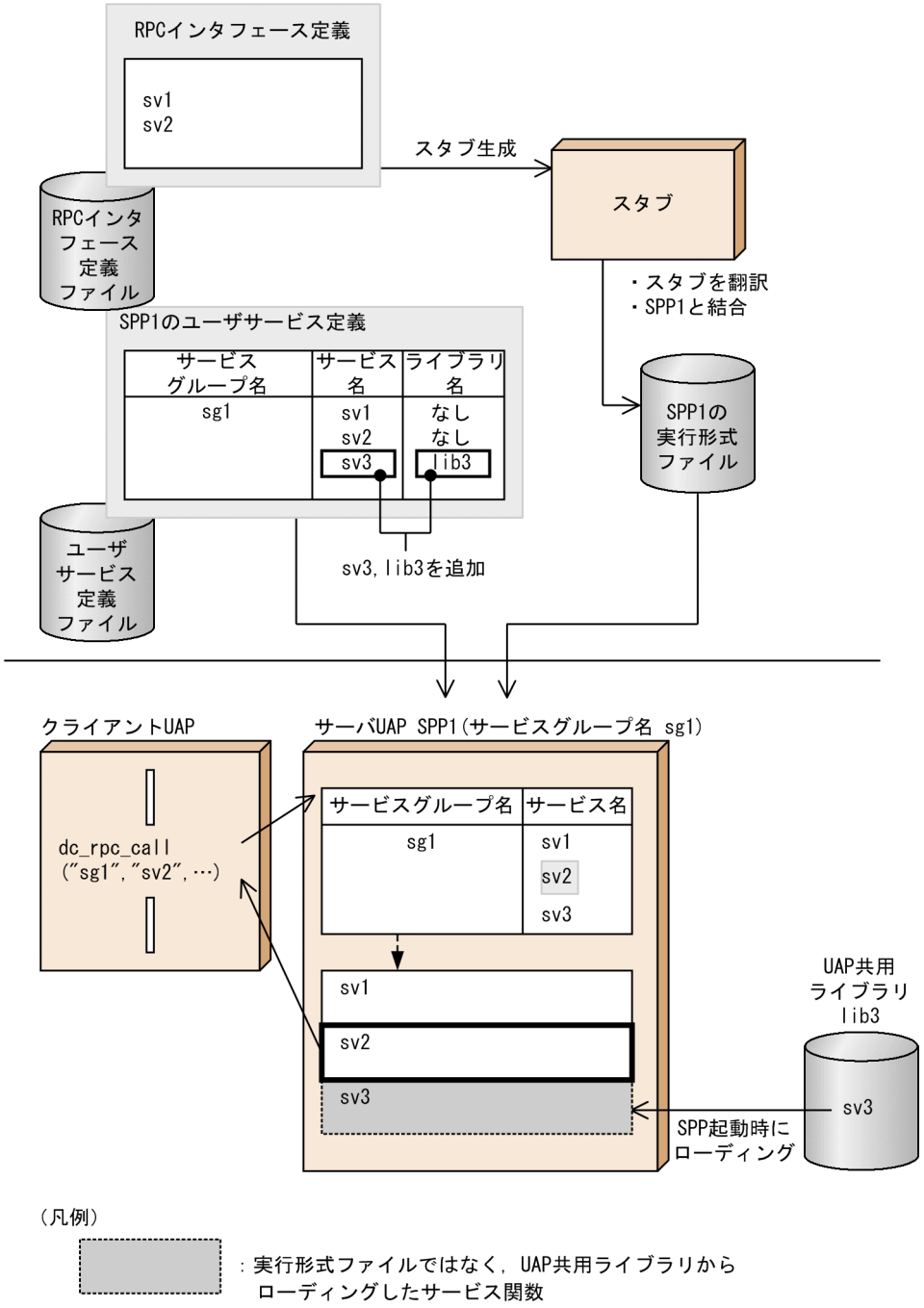
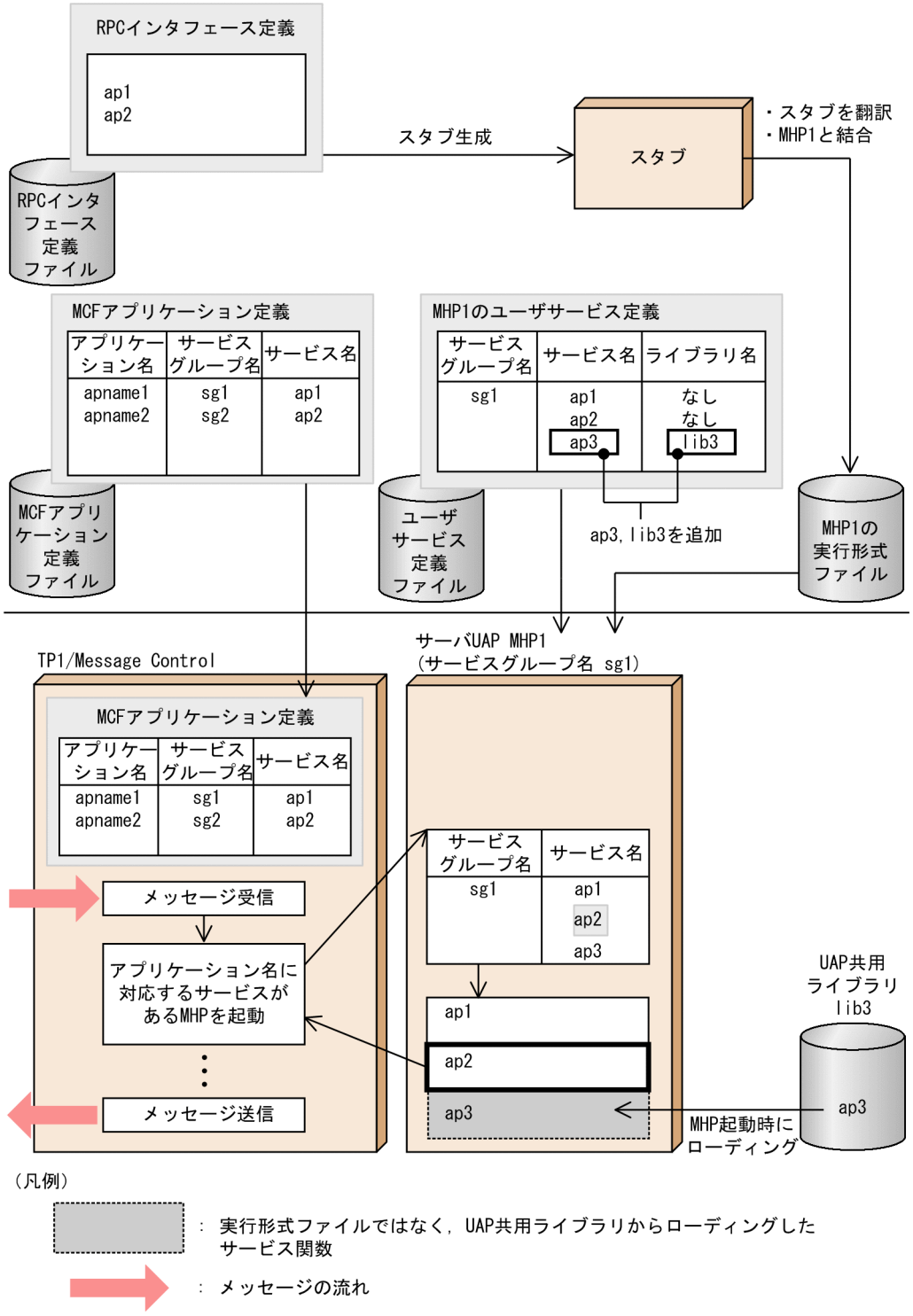


図 2-23　サービス関数動的ローディング機能とスタブを併用する場合（MHP）

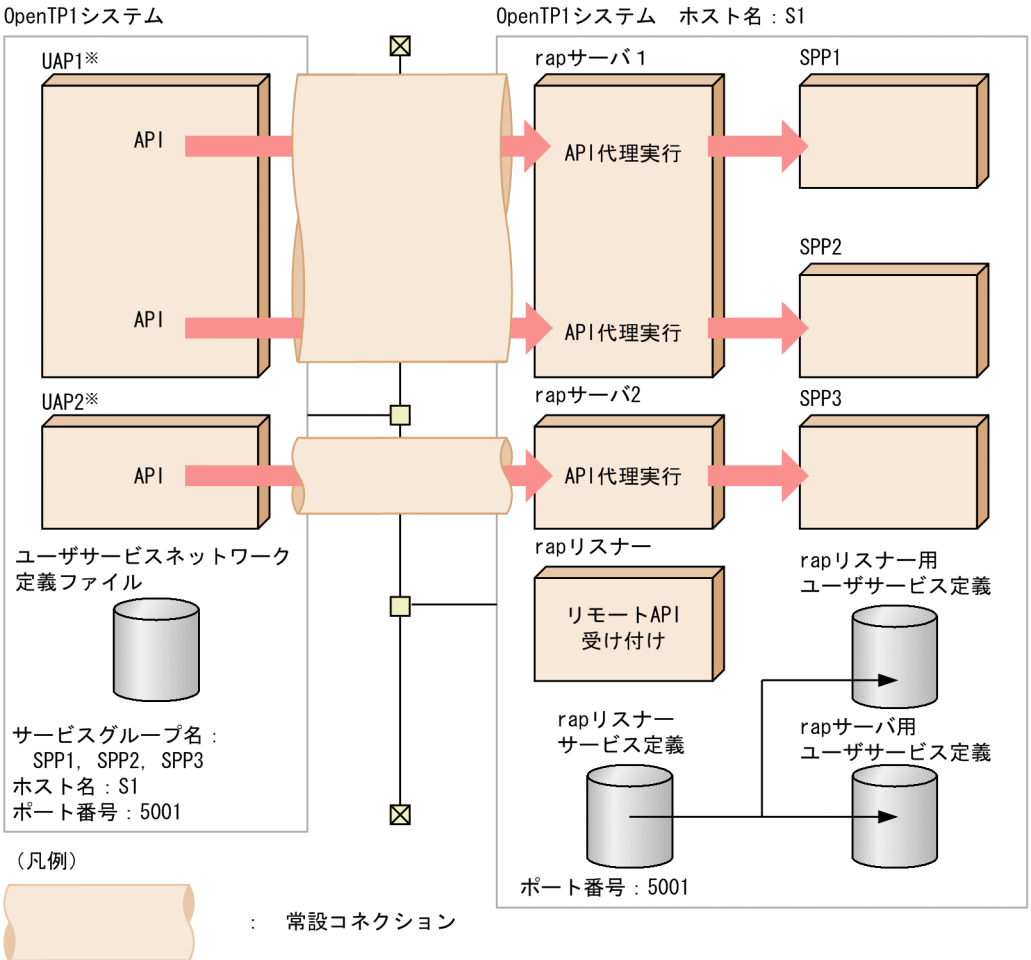


2.2　リモート API 機能

OpenTP1 では，クライアント側のノードにある UAP が発行した API を，OpenTP1 がサーバ側に転送してサーバ側のプロセスで実行できます。このような機能を**リモート API 機能**といいます。リモート API 機能を要求するクライアント側のノードにある UAP を **rap クライアント**といいます。rap クライアントが発行した API を，OpenTP1 の **rap リスナー**が受け付け，**rap サーバ**がサーバ側のノードで実行します。rap リスナー，rap サーバは OpenTP1 のユーザサービスとして動作します。ユーザは rapsetup コマンドで rap リスナーと rap サーバの動作環境を設定してください。

リモート API 機能を使用するユーザは，通信相手のサービス情報（ホスト名とポート番号）を，ユーザサービスネットワーク定義に-w オプション付きで定義しておきます。また，サーバ側に rap リスナーサービス定義を作成し，rapdfgen コマンドで rap リスナー用ユーザサービス定義と rap サーバ用ユーザサービス定義を自動生成します。リモート API 機能を次の図に示します。

図 2-24　リモート API 機能



注　　リモートAPI機能では，XATMIインタフェースを使用してはいけません。
　　XATMIインタフェースを使用した場合，OpenTP1の動作は保証されません。
注※　リモートAPI機能を使用するUAPはユーザサービス定義の
　　rpc_destination_modeオペランドでdefinitionを指定します。

次に，リモート API 機能で代理実行できる API を rap クライアントの種類ごとに示します。

- TP1/Server Base，TP1/LiNK が rap クライアントとなる場合

C 言語ライブラリ	COBOL-UAP 作成用プログラム
dc_rpc_call	CBLDCRPC('CALL ')

TP1/LiNK を使用する場合は，マニュアル「TP1/LiNK 使用の手引」を参照してください。

- TP1/Client/P，TP1/Client/W が rap クライアントとなる場合

C 言語ライブラリ	COBOL-UAP 作成用プログラム
dc_rpc_call_s	CBLDCRPS（'CALL '）
dc_trn_begin_s	CBLDCTRS（'BEGIN '）
dc_trn_chained_commit_s	CBLDCTRS（'C-COMMIT'）
dc_trn_chained_rollback_s	CBLDCTRS（'C-ROLL '）
dc_trn_unchained_commit_s	CBLDCTRS（'U-COMMIT'）
dc_trn_unchained_rollback_s	CBLDCTRS（'U-ROLL '）

TP1/Client/P および TP1/Client/W を使用する場合は，マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」を参照してください。

- TP1/Client/J が rap クライアントとなる場合

メソッド
rpcCall
trnBegin
trnChainedCommit
trnChainedRollback
TrnUnchainedCommit
trnUnchainedRollback

TP1/Client/J 編を使用する場合は，マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/J 編」を参照してください。

- TP1/Client for .NET Framework が rap クライアントとなる場合

メソッド
Call
Begin
CommitChained
RollbackChained
Commit

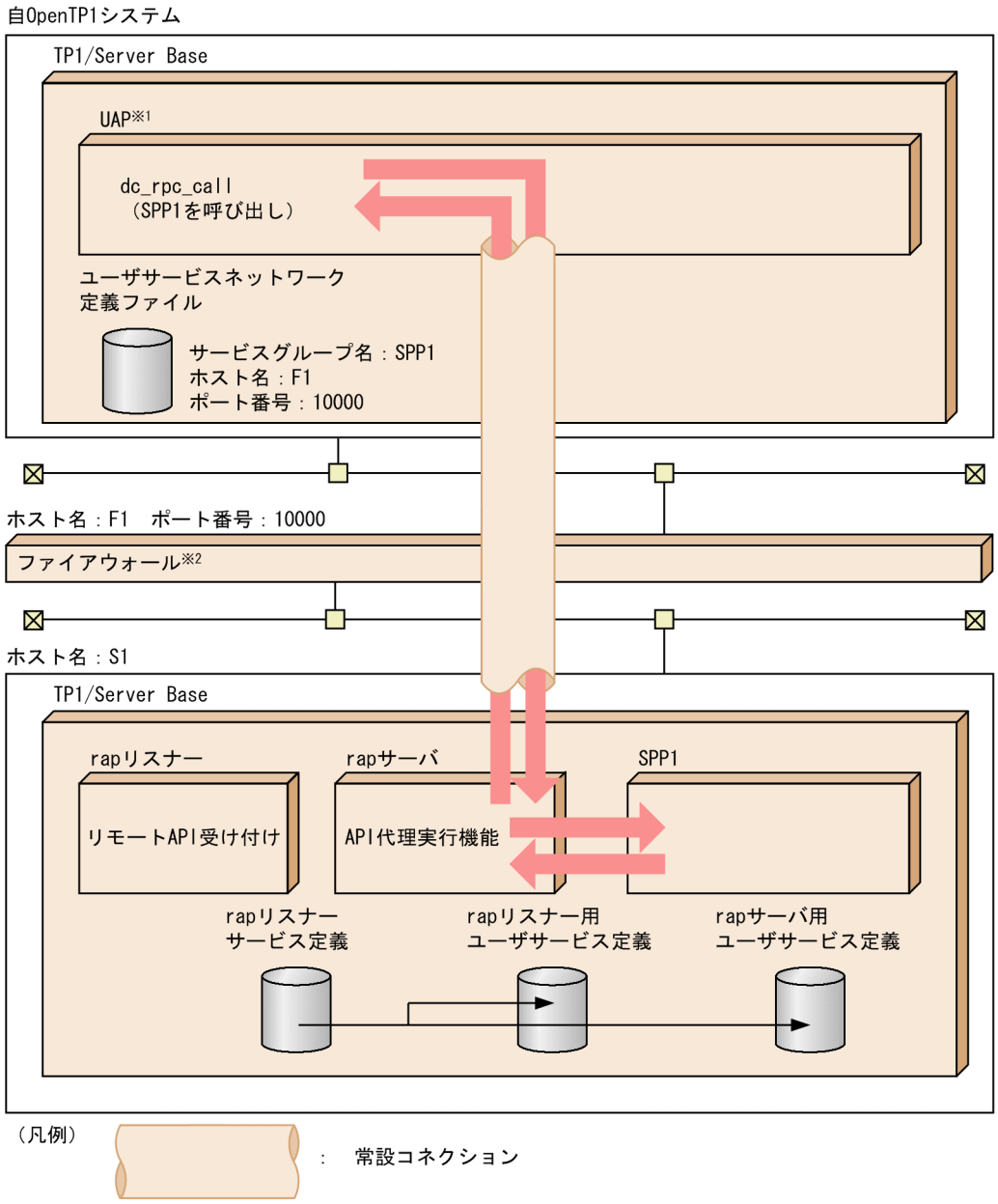
メソッド
Rollback

TP1/Client for .NET Framework を使用する場合は，マニュアル「TP1/Client for .NET Framework 使用の手引」を参照してください。

2.2.1 リモート API 機能の使用例

リモート API 機能を使うと，ファイアウォールの内側にある UAP に対してもサービスを要求できます。ファイアウォールの内側への RPC を次の図に示します。

図 2-25　ファイアウォールの内側にある UAP へのリモートプロシジャコール



注1　ファイアウォールを通過する場合は，次の点に注意してください。
・非同期応答型RPCを使用してはいけません。
・ファイアウォールを通過した場合，通過先のUAP（この図の場合 SPP1）は，トランザクション ブランチになりません。
注2　リモートAPI機能では，XATMIインタフェースを使用してはいけません。
XATMIインタフェースを使用した場合，OpenTP1の動作は保証されません。
注※1　UAPはユーザサービス定義のrpc_destination_modeオペランドで，definitionを指定します。
注※2　ファイアウォールにはGauntletを想定しています。

2.2.2　常設コネクション

OpenTP1 は，リモート API を要求した UAP（rap クライアント）と rap サーバとの間に，論理的な通信路（**常設コネクション**）を設定します。

常設コネクションのスケジュール方法には，スタティックコネクションスケジュールモードとダイナミックコネクションスケジュールモードがあります。どちらを使用するかは，rap リスナーサービス定義の rap_connection_assign_type オペランドで指定できます。

2.2.3　コネクトモード

常設コネクションの管理方法は，コネクションの確立・解放方法によって二つに分けられます。コネクションの確立，解放を OpenTP1 が管理する形態を**オートコネクトモード**，ユーザが管理する形態を**非オートコネクトモード**といいます。ユーザは常設コネクションをオートコネクトモードで管理するか，非オートコネクトモードで管理するか，rap クライアントのユーザサービス定義で定義します。

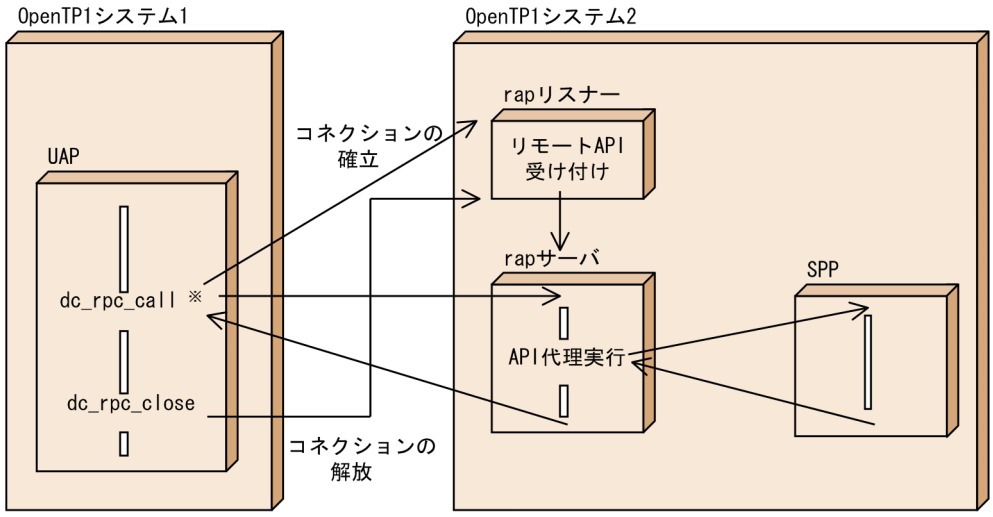
(1)　オートコネクトモード

OpenTP1 が常設コネクションの確立・解放を管理する形態です。rap クライアントが，ユーザサービスネットワーク定義に-w オプション付きで定義されているサービスグループ名を指定して dc_rpc_call 関数を発行した場合に，自動的に常設コネクションを確立します。

ユーザサービスネットワーク定義に定義されているサービスグループに dc_rpc_call 関数を発行した時点から，dc_rpc_close 関数を発行して RPC を終了するまで常設コネクションを確保します。

オートコネクトモードの概要を次の図に示します。

図 2-26　オートコネクトモードの概要



注　　OpenTP1が常設コネクションの確立と解放を管理しています。
注※　rapクライアントのユーザサービスネットワーク定義に-wオプションが定義されています。

rap クライアントは rap サーバとの間で設定するコネクション数に上限があり，dc_rpc_call 関数の呼び出し時にこのコネクション数の上限を超える場合は，rap クライアントプロセスで使用しているコネクションの中で，最も以前に使われたコネクションを自動的に解放したあと，新たにコネクションを確立します。

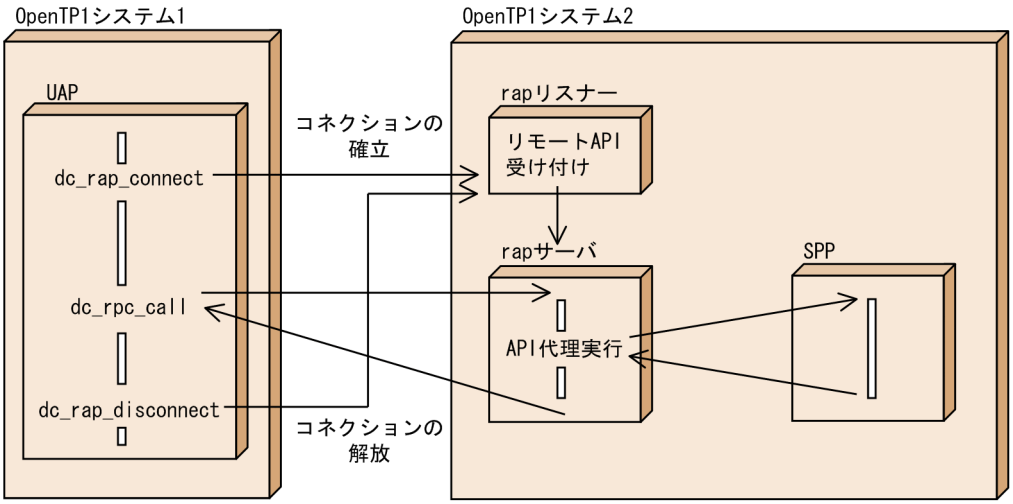
ただし，連鎖 RPC を実行中のコネクションは，解放の対象とはなりません。このことが原因で解放できるコネクションがない場合は，API を発行した UAP はダウンします。

(2) 非オートコネクトモード

ユーザが常設コネクションの確立・解放を管理する形態です。ユーザは rap クライアントでコネクションを確立するために dc_rap_connect 関数【CBLDCRAP('CONNECT ')】を発行し，解放するために dc_rap_disconnect 関数【CBLDCRAP('DISCNCT ')】を発行します。rap クライアントが，ユーザサービスネットワーク定義に-w オプション付きで定義されているサービスグループ名を指定して dc_rpc_call 関数を発行したときに，ユーザが常設コネクションを確立していない場合，dc_rpc_call 関数はエラーリターンします。リターンコードは DCRPCER_PROTO です。

非オートコネクトモードの概要を次の図に示します。

図 2-27 非オートコネクトモードの概要



dc_rap_connect 関数発行時に，rap クライアントは rap サーバとの間で設定するコネクション数の上限を超える場合，エラーリターンします。

2.2.4 常設コネクションでの連鎖 RPC

ここでは，常設コネクションでの連鎖 RPC として，オートコネクトモードでの連鎖 RPC，および非オートコネクトモードでの連鎖 RPC について説明します。

(1) オートコネクトモードでの連鎖 RPC

rap クライアントは rap サーバとの間で設定するコネクション数に上限があり，dc_rpc_call 関数の呼び出し時にこのコネクション数の上限を超える場合は，rap クライアントプロセスで使用しているコネクションの中で，最も以前に使われたコネクションを自動的に解放したあと，新たにコネクションを確立します。

ただし，連鎖 RPC を実行中のコネクションは，解放の対象とはなりません。このことが原因で解放できるコネクションがない場合は，API を発行した UAP はダウンします。

(2) 非オートコネクトモードでの連鎖 RPC

連鎖 RPC 実行中に dc_rap_disconnect 関数が呼び出されたり，API を発行した UAP のダウンや通信障害が発生した場合，API を代理実行中の rap サーバは，連鎖 RPC を実行している SPP とのコネクションやステータスをリセットするために，いったん異常終了し，再起動します。

2.2.5 注意事項

リモート API 機能を利用する場合は，次の点に注意してください。

- 非同期応答型 RPC を使用してはいけません。非同期応答型 RPC を使用した場合，リモート API 機能として動作しないで，通常の RPC として動作します。
- リモート API 機能では，XATMI インタフェースを使用してはいけません。XATMI インタフェースを使用した場合，OpenTP1 の動作は保証されません。
- トランザクション内から，リモート API 機能を使って dc_rpc_call を発行しても，トランザクションとして処理されません。
- リモート API 機能による通信は RPC トレースに取得されません。ただし，rap サーバで代理実行した dc_rpc_call は RPC トレースに取得されます。
- レスポンス統計情報，および通信遅延時間統計情報には，リモート API 機能による通信部分は取得されません。
- アプリケーションゲートウェイ型ファイアウォールの，ゲートウェイを介して RPC を実行する場合などで，ユーザサービスネットワーク定義の dcsvgdef 定義コマンドに-w オプションを指定してリモート API 機能を使用するときは，トランザクション属性で dc_rpc_call 関数を呼び出してもトランザクションにはなりません。そのため，リモート API 機能を使用した場合，トランザクション内から連鎖 RPC を開始して，同期点処理で連鎖 RPC を終了させる運用は正しく動作しません。flags に DCNOFLAGS を指定した dc_rpc_call 関数を呼び出して，明示的に連鎖 RPC を終了してください。
- 通常，rap サーバは rap リスナーによって自動的に開始されます。rap サーバを単独で終了（dcsvstop コマンド実行）または開始（dcsvstart コマンド実行）しないでください。ただし，次の場合は，dcsvstart コマンドで rap サーバを開始してください。

  定義エラーなどが原因で，rap サーバの開始に失敗した場合

  定義エラーなどが原因で rap サーバを開始できない場合でも，rap リスナーは rap サーバを開始できないことを検知できません。そのため，システムは，リモート API サービスの準備中（KFCA26950-I メッセージが出力された状態）のままになってしまいます。rap リスナーによる rap サーバの開始時に，要因コードが CONFIGURATION の KFCA01812-E メッセージが出力された場合は，rap サーバの定義を見直し，dcsvstart コマンドで rap サーバを開始してください。なお，rap サーバの定義エラーは，KFCA00244-E メッセージでは検出できません。

rap リスナーおよび rap サーバの終了中に rap リスナーがダウンした場合

rap リスナーのダウン後に，dcsvstart コマンドで rap リスナーを開始しても，rap サーバが KFCA26950-I メッセージを出力し，リモート API サービスの準備中のままになることがあります。rap サーバが開始されないときは，dcsvstart コマンドで rap サーバを開始してください。

- rap サーバのオンライン中に，rap サーバに対して scdhold コマンドを実行しないでください。
- rap サーバと同一のノードにある UAP に対して，リモート API 機能を使用したサービス要求をしないでください。サービス要求をした場合の動作は保証しません。
- 非オートコネクトモードで，コネクション確立後に発行した dc_rpc_call がエラーとなった場合は，dc_rap_disconnect でコネクションを解放したあと，再度 dc_rap_connect でコネクションを確立させてください。

2.3　トランザクション制御

OpenTP1 では，クライアント／サーバ形態の通信の**トランザクション制御**ができます。これによって，複数のプロセスにわたる UAP の処理を一つのトランザクションとして処理できます。OpenTP1 の UAP で使えるトランザクション制御の関数には，次に示す 2 種類があります。

- OpenTP1 独自のインタフェース
- TX インタフェース（X/Open の仕様に準拠したトランザクション制御）

ここでは，**OpenTP1 独自のインタフェース**について説明します。TX インタフェースについては，「5.2 TX インタフェース（トランザクション制御）」を参照してください。

ここでは，**クライアント／サーバ形態の UAP**（SUP，SPP）のトランザクション制御について説明します。メッセージ送受信形態の UAP（MHP）のトランザクション制御については，「3.7　MCF のトランザクション制御」を参照してください。

- TP1/LiNK の場合の注意

  TP1/LiNK で使う UAP でトランザクション制御をする場合は，TP1/LiNK の実行環境を設定するときに，トランザクション機能を使う指定をしてください。

2.3.1　クライアント／サーバ形態の通信のトランザクション

OpenTP1 では，RPC を使用して複数の UAP プロセスにわたっている処理を，1 件のトランザクション処理にできます。このようなトランザクションを**グローバルトランザクション**といいます。このグローバルトランザクションによって，クライアント／サーバ形態のトランザクションができるようになります。

(1)　トランザクションの開始と同期点の取得

クライアント／サーバ形態の通信でトランザクションを制御するときは，**トランザクションの開始**と**同期点の取得**を UAP で明示します。

トランザクションを開始するときは，次に示す関数を呼び出します。

- dc_trn_begin 関数【CBLDCTRN('BEGIN ')】

同期点を取得するときは，次に示す関数を呼び出します。

- dc_trn_chained_commit 関数【CBLDCTRN('C-COMMIT')】
- dc_trn_chained_rollback 関数【CBLDCTRN('C-ROLL ')】
- dc_trn_unchained_commit 関数【CBLDCTRN('U-COMMIT')】
- dc_trn_unchained_rollback 関数【CBLDCTRN('U-ROLL ')】

トランザクションを開始した時点で，クライアント UAP はルートトランザクションブランチになります。トランザクションの同期点取得（コミット）は，トランザクションを開始したルートトランザクションブランチで呼び出します。

トランザクションを開始したあとに，そのグローバルトランザクションの中で新しいトランザクションは開始できません。

トランザクションとして実行中の UAP から要求されたサービスは，要求された時点ですでにトランザクションとして実行しています。このとき，要求されたサービスの処理から dc_trn_begin 関数は呼び出せません。

(2) トランザクションを制御する関数を使える UAP

トランザクションの開始，または同期点を取得する関数を呼び出せるのは，SUP と SPP です。MHP のトランザクション処理は OpenTP1 で自動的に制御しているので，MHP からトランザクションを制御する関数は呼び出せません。また，MHP から dc_rpc_call 関数でサービスを要求される SPP でも，トランザクションを制御する関数は呼び出せません。

オフラインの業務をする UAP では，トランザクションを制御する関数を使えません。

OpenTP1 クライアント機能の UAP（CUP）は，TP1/Client のライブラリにあるトランザクション制御の関数を使います。

2.3.2 同期点の取得

グローバルトランザクションを構成するトランザクションブランチのすべてを，同期を取って同じ決着結果（**コミット**または**ロールバック**）で終了させることを**同期点の取得**といいます。

(1) コミットの関数の呼び出し

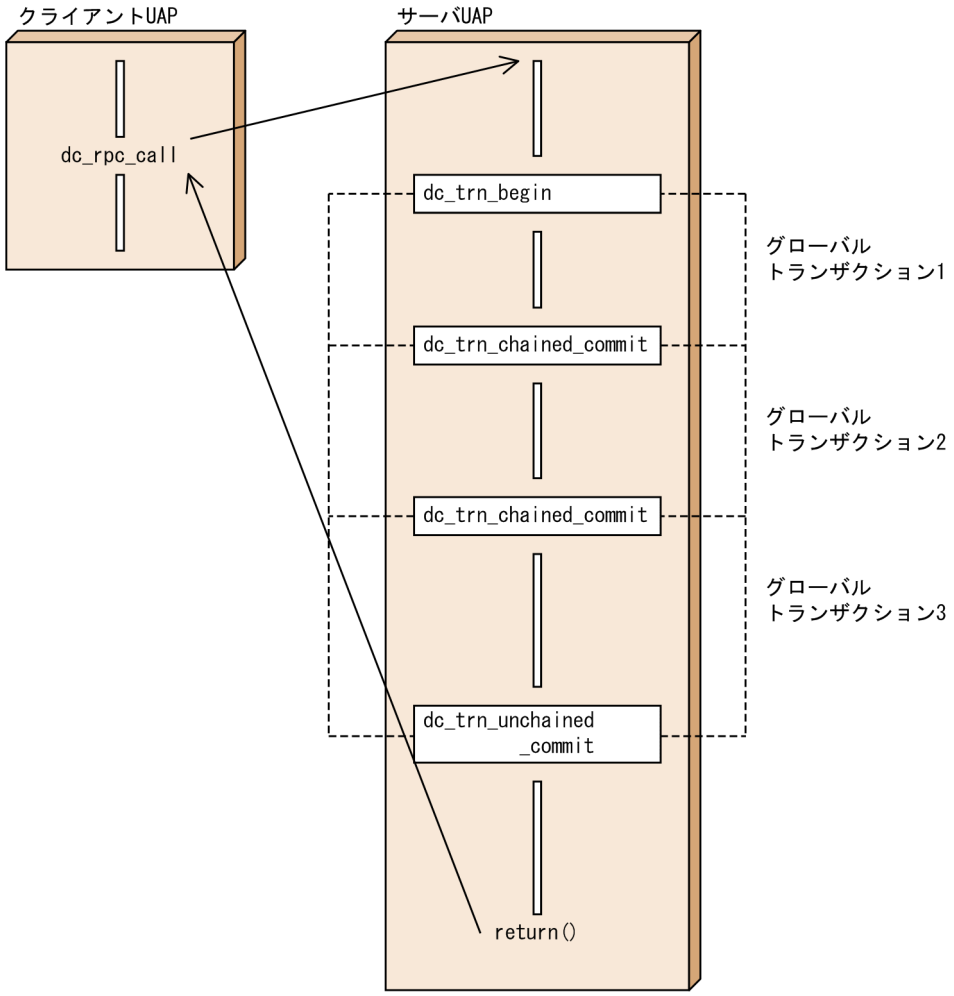
UAP からコミットを指示する関数は，**ルートトランザクションブランチ**（dc_trn_begin 関数でトランザクションを開始した SPP または SUP）からだけ使えます。ルートトランザクションブランチ以外のトランザクションブランチでは，コミットの関数は使えません。グローバルトランザクションは，すべてのトランザクションブランチが正常に終了したことで正常終了となります。

(a) 連鎖／非連鎖モードのコミット

トランザクション処理の同期点を取得する関数には，一つのトランザクションの終了後，同期点を取得して次のトランザクションを続けて起動する**連鎖モードのコミット**（dc_trn_chained_commit 関数）と，トランザクションの終了で同期点を取得したあとに，新たなトランザクションを起動しない**非連鎖モードのコミット**（dc_trn_unchained_commit 関数）があります。

トランザクションの連鎖／非連鎖モードを次の図に示します。

図 2-28　トランザクションの連鎖／非連鎖モード



(2)　ロールバックの関数の呼び出し

トランザクションを UAP の判断でロールバックとしたいときは，UAP からロールバックを要求できます。

(a)　連鎖／非連鎖モードのロールバック

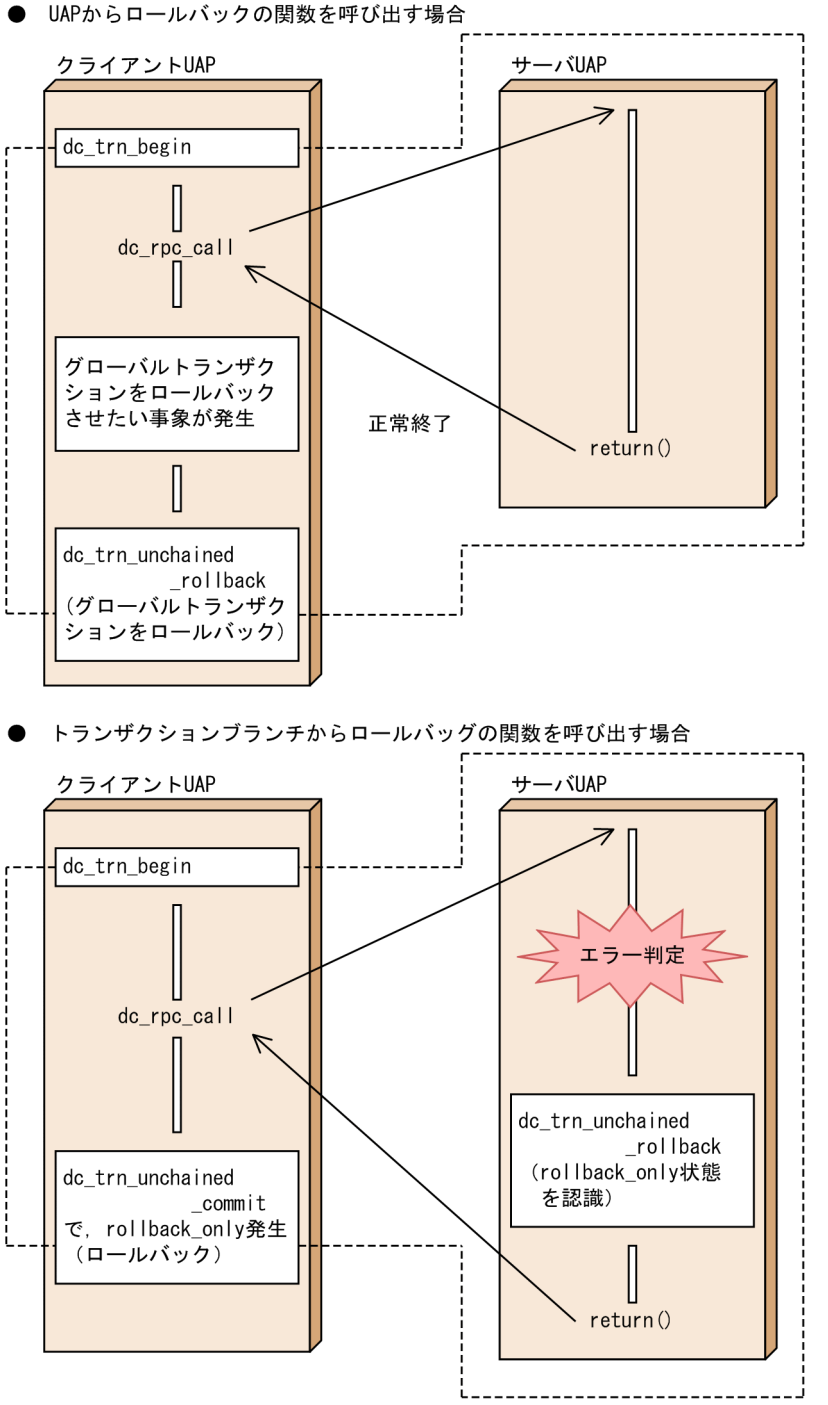
ロールバックの関数には，**連鎖モードのロールバック**（dc_trn_chained_rollback 関数）と**非連鎖モードのロールバック**（dc_trn_unchained_rollback 関数）があります。連鎖モードのロールバックの場合は，ロールバック処理後も新しいグローバルトランザクションの範囲にあります。非連鎖モードのロールバックをルートトランザクションブランチから呼び出した場合は，グローバルトランザクションの範囲にはありません。

連鎖モードのロールバックは，ルートトランザクションブランチからしか呼び出せません。非連鎖モードのロールバックは，どのトランザクションブランチからでも呼び出せます。

ルート以外のトランザクションブランチから非連鎖モードのロールバックを呼び出した場合は，そのトランザクションブランチがロールバック対象である（**rollback_only 状態**）ことをルートトランザクションブランチに通知します。この場合，ロールバック関数のあとから，dc_rpc_call 関数がリターンするまでの処理もグローバルトランザクションの範囲にあります。

トランザクションのロールバックを次の図に示します。

図 2-29　トランザクションのロールバック



注　　は，ロールバックの関数でロールバックされる処理の範囲を示します。

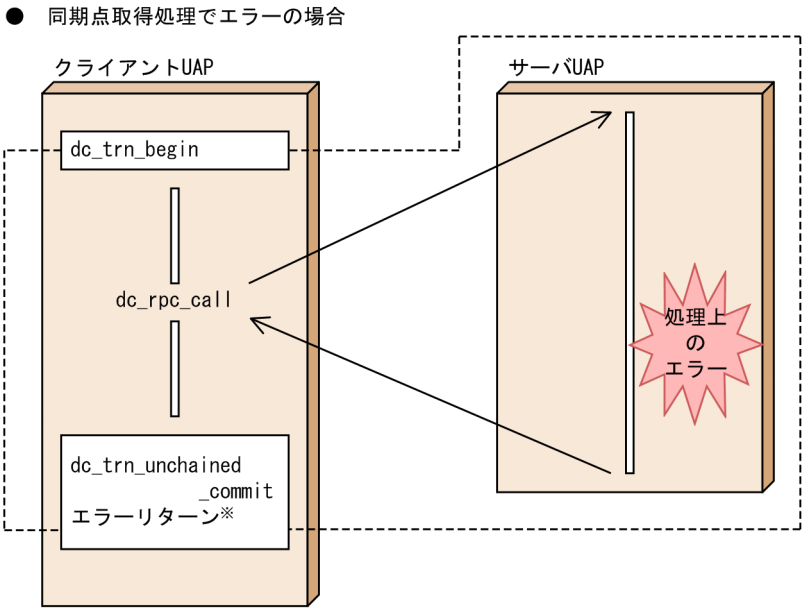
(3)　同期点を取得する処理でエラーが起こった場合

トランザクションの同期点を取得する処理でエラーが起こった場合，そのトランザクションが同期点の 1 相目まで完了していればコミットして，1 相目まで完了していなければロールバックします。グローバル

トランザクション内のどれか一つのトランザクションブランチがロールバックになった場合は，グローバルトランザクション全体がロールバックになります。

同期点を取得する処理でエラーが起こった場合のロールバックを次の図に示します。

図 2-30　同期点を取得する処理でエラーが起こった場合のロールバック



注※　トランザクションが同期点の1相目まで完了していれば，　内の処理をコミットして，1相目まで完了していなければロールバックします。

(4)　同期点を取得する関数を呼び出さなかった場合の処理

同期点を取得する関数を呼び出す前に UAP が異常終了した場合は，UAP の同期点の決着結果はロールバックになります。

同期点を取得する関数を呼び出さないで，ルートトランザクションブランチの UAP が exit() した場合は，OpenTP1 が自動的にコミットの処理をします。そのコミット処理で 1 相目が完了する前にエラーが起こった場合は，グローバルトランザクションはロールバックされます。この場合，UAP には通知できません。

2.3.3　トランザクション属性の指定

UAP の実行環境を設定するときには，UAP のプロセスをトランザクションとして稼働させるかどうかを指定しておきます。トランザクションとして稼働する指定をした UAP プロセスを，**トランザクション属性の UAP** といいます。ファイルの更新など，トランザクション処理をする UAP には，トランザクション属性である指定をしておく必要があります。

(1)　UAP をトランザクション属性とする方法

サーバ UAP のプロセスをトランザクションブランチとしたい場合は，**トランザクション属性の UAP** であることを指定します。トランザクション属性を指定する方法を次に示します。

- TP1/Server Base の場合：
  ユーザサービス定義の atomic_update オペランドに Y を指定
- TP1/LiNK の場合：
  ユーザサーバにトランザクション機能を使うことを指定

トランザクション属性を指定した UAP の処理がトランザクションとなるのは，次の場合です。

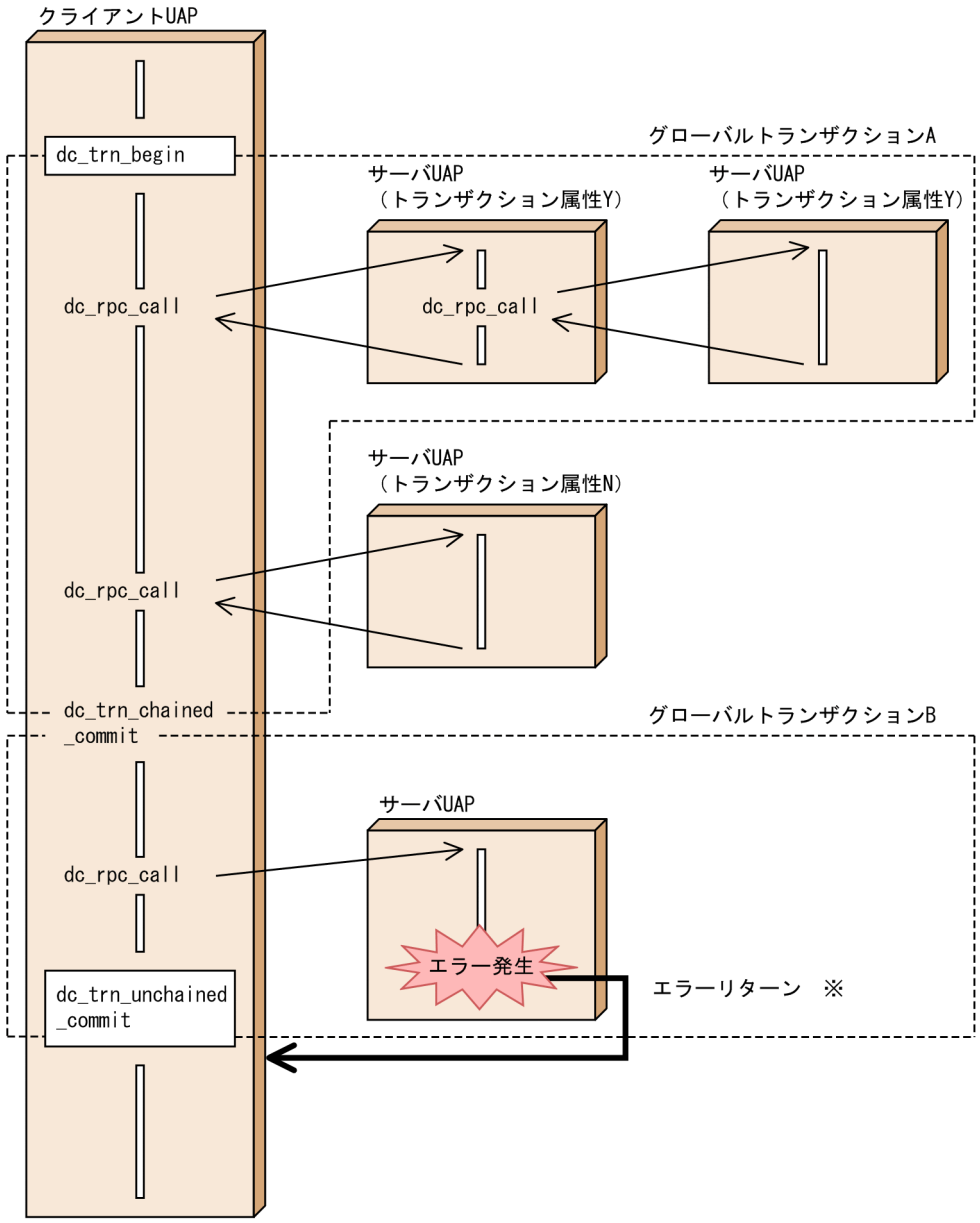
- トランザクション属性を指定した UAP から，トランザクションの開始（dc_trn_begin 関数）を呼び出して正常にリターンした場合
- トランザクションとして処理しているほかの UAP から，dc_rpc_call 関数でサービスを要求された場合

(2) UAP をトランザクション属性なしとする場合

演算だけのサーバ UAP など，処理をトランザクションとして保証しなくてもよいサーバ UAP には，**トランザクション属性なし**（atomic_update オペランドに N，またはトランザクション機能を使わないと指定）にしておきます。トランザクション属性なしと指定したサーバ UAP は，グローバルトランザクションの処理とは無関係に，いつでもほかのクライアント UAP にサービスを提供できます。そのため，複数のクライアント UAP からサービスを要求された場合でも，同期点を取得する処理が完了するのを待たないで並行して処理できて，サービス要求待ちのオーバヘッドを減らせます。

RPC とトランザクション属性の関係を次の図に示します。

図 2-31　RPC とトランザクション属性の関係



注※

グローバルトランザクション B でアクセスした資源の内容は，グローバルトランザクション B が開始する直前の状態に戻ります。グローバルトランザクション B がロールバックしたことは，同期点を取得する関数（この図の場合は dc_trn_unchained_commit 関数）がエラーリターンすることで通知されます。

(a)　トランザクションの処理から非トランザクショナル RPC の発行

トランザクションの処理からサービスを要求した場合に，サービスを要求された UAP がトランザクション属性のときは，トランザクションの処理となります。このようなサービス要求を，トランザクション処理としないこともできます。トランザクション処理としたくない場合は，dc_rpc_call 関数の引数にトランザクションでない RPC であることを指定します。

2.3.4　リモートプロシジャコールの形態と同期点の関係

トランザクションとして稼働しているUAP（SUP，SPP，MHP）からRPCで呼ばれたSPPにトランザクション属性が指定してある場合は，そのSPPはトランザクションとして稼働します。各トランザクションブランチは，一つのグローバルトランザクションとして同期を取れます。したがって，各サーバUAPのプロセスは，処理終了後，dc_rpc_call関数を呼び出したUAPにリターンしても，ルートトランザクションブランチに戻って同期点処理が完了するまでは，次のサービス要求を受け付けられません。また，サーバUAPで確保している資源も解放されません。これは非同期応答型RPC，非応答型RPC，連鎖RPCの場合でも同様です。

このように，UAPの処理ではRPCのトランザクション制御で，複数のUAPで同期が取れます。

クライアントUAPの同期点処理が完了する前に，サーバUAPのプロセスでほかのサービス要求を処理できる場合があります。これを**トランザクションの最適化**といいます。トランザクションの最適化については，「2.3.5　トランザクションの最適化」を参照してください。

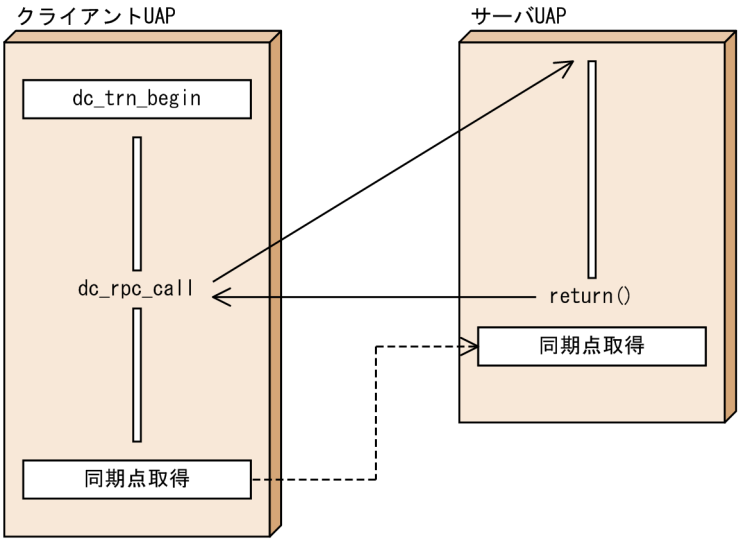
(1)　同期応答型RPCと同期点の関係

同期応答型RPCのトランザクション処理の場合は，ルートトランザクションブランチに処理結果が戻って，同期点処理を終えた時点でトランザクションの終了となります。

トランザクションを最適化する条件がそろっている場合，サーバUAPのプロセスは処理が終了した時点で，次のサービス要求を受け付けられます。

同期応答型RPCと同期点の関係を次の図に示します。

図 2-32　同期応答型RPCと同期点の関係

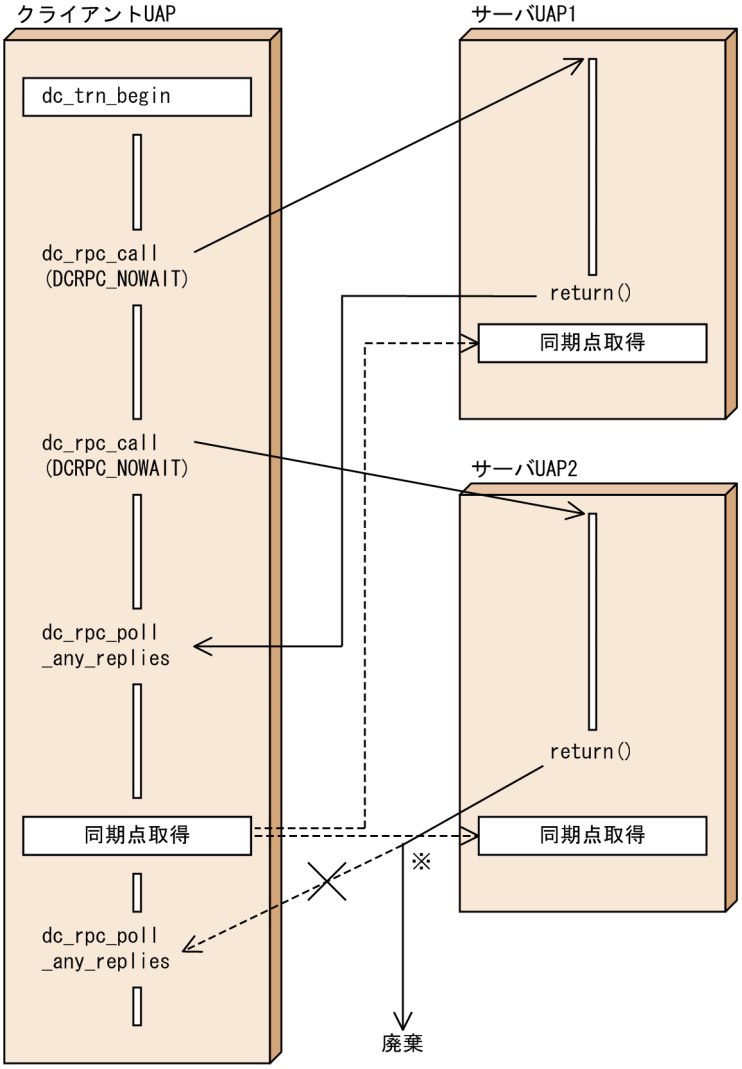


(2) 非同期応答型 RPC と同期点の関係

非同期応答型 RPC のトランザクション処理の場合は，クライアント UAP で同期点処理を終えた時点で RPC の処理を終了とします。同期点処理後にサーバ UAP から応答が返ってきても，dc_rpc_call 関数を呼び出した UAP では受信できません。

非同期応答型 RPC と同期点の関係を次の図に示します。

図 2-33 非同期応答型 RPC と同期点の関係



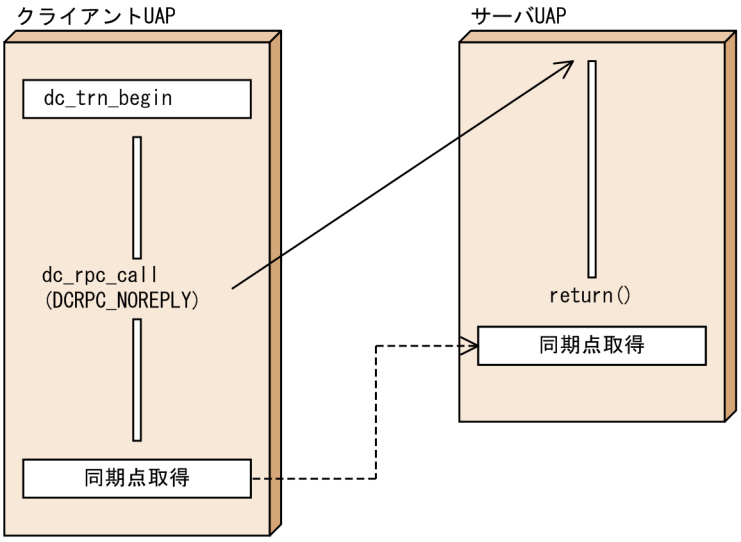
注※ クライアントUAPで同期点取得処理が完了したあとにdc_rpc_poll_any_replies関数を呼び出しても，応答は受信できません。
非同期応答型RPCで要求したサービスの結果をすべて受信する場合は，dc_rpc_call関数を呼び出した数だけ，dc_rpc_poll_any_replies関数を，必ず同期点取得処理前に呼び出してください。また，応答の受信を拒否する場合は，dc_rpc_discard_further_replies関数を呼び出してください。

(3) 非応答型 RPC と同期点の関係

非応答型 RPC のトランザクション処理の場合は，クライアント UAP の同期点取得でサーバ UAP の処理終了を待って，そのあとで同期点処理をします。

非応答型 RPC と同期点の関係を次の図に示します。

図 2-34　非応答型 RPC と同期点の関係



(4)　連鎖 RPC と同期点の関係

連鎖 RPC を使った場合は，一つのサーバ UAP のプロセスで実行します。したがって，トランザクションブランチも連鎖 RPC を使った回数に関係なく，一つになります。

連鎖 RPC のトランザクション処理の場合，同期点処理を終了した時点でトランザクションが終了して，処理していたサーバ UAP のプロセスは解放されます。

トランザクション実行中に非トランザクションの連鎖 RPC を使った場合，通常は同期点処理を終了した時点で，処理していたサーバ UAP のプロセスを解放します。同期点処理の終了で処理していたサーバ UAP のプロセスを解放しないで，同期応答型 RPC（flags に DCNOFLAGS を指定）で解放する場合は，ユーザサービス定義の rpc_extend_function オペランドに 00000002 を指定してください。

同期応答型 RPC で連鎖 RPC を終了した場合，トランザクションを最適化する条件がそろっているときは，サーバ UAP のプロセスは処理が終了した時点で，次のサービス要求を受け付けられます。

連鎖 RPC と同期点の関係を以降の図に示します。

図 2-35　連鎖 RPC と同期点の関係（トランザクションの連鎖 RPC）

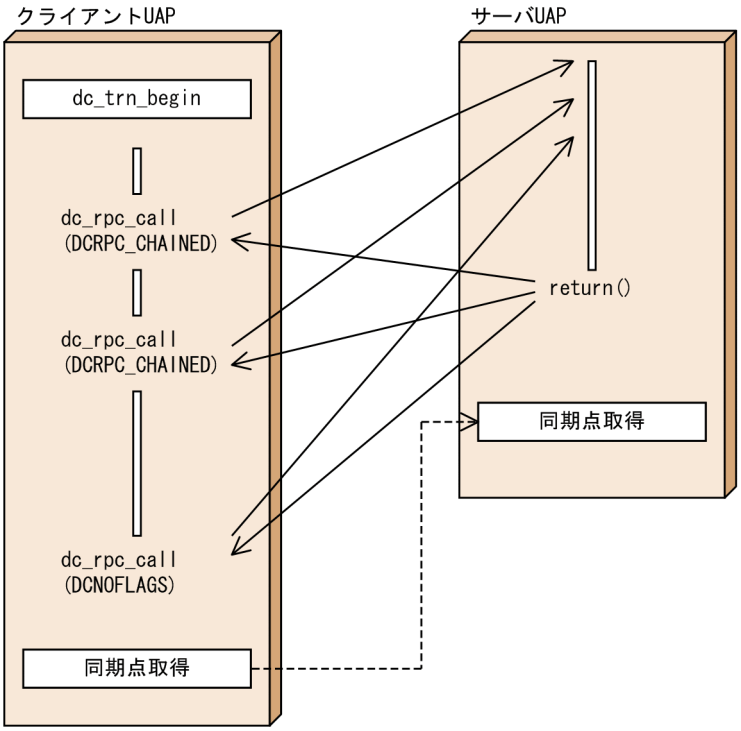
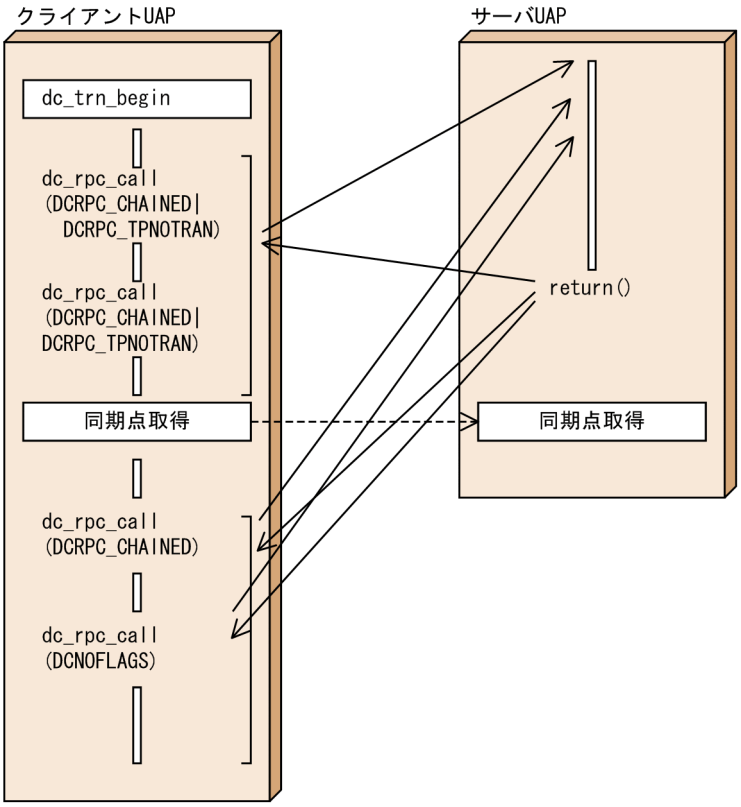


図 2-36　連鎖 RPC と同期点の関係（非トランザクションの連鎖 RPC で終了しない指定をした場合）



(5) リモートプロシジャコールのエラーリターン値と同期点

dc_rpc_call 関数，または dc_rpc_poll_any_replies 関数がエラーリターンしても，トランザクションの同期点がコミットとなる場合があります。

また，リターン値によっては，必ずトランザクションをロールバックさせなければならない場合があります。この場合は，ロールバックの関数（dc_trn_chained_rollback 関数，または dc_trn_unchained_rollback 関数）でロールバックさせてください。

必ずロールバックさせなければならない dc_rpc_call 関数，または dc_rpc_poll_any_replies 関数のリターン値については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編の説明を参照してください。

2.3.5 トランザクションの最適化

OpenTP1 では，トランザクション処理の性能を上げるため，次のような最適化の処理をしています。

- コミット最適化　1.
- プリペア最適化　2.
- 非同期プリペア最適化　3.
- 1 相最適化　4.
- リードオンリー最適化　5.
- ノーアクセス最適化　6.
- ロールバック最適化　7.

それぞれの最適化には，最適化できる条件があります。条件を満たした UAP を作成すると，トランザクション処理の性能を上げることができます。

最適化の優先順位を次に示します。

5.6.7.> 2.> 3.> 4.（1.はほかの最適化と同時に実行されます）

トランザクションの最適化は，クライアント側のトランザクションブランチとサーバ側のトランザクションブランチとの間で，同期点処理の効率を上げることを目的としています。そのため，一つのグローバルトランザクション内で複数の最適化を混在できます。

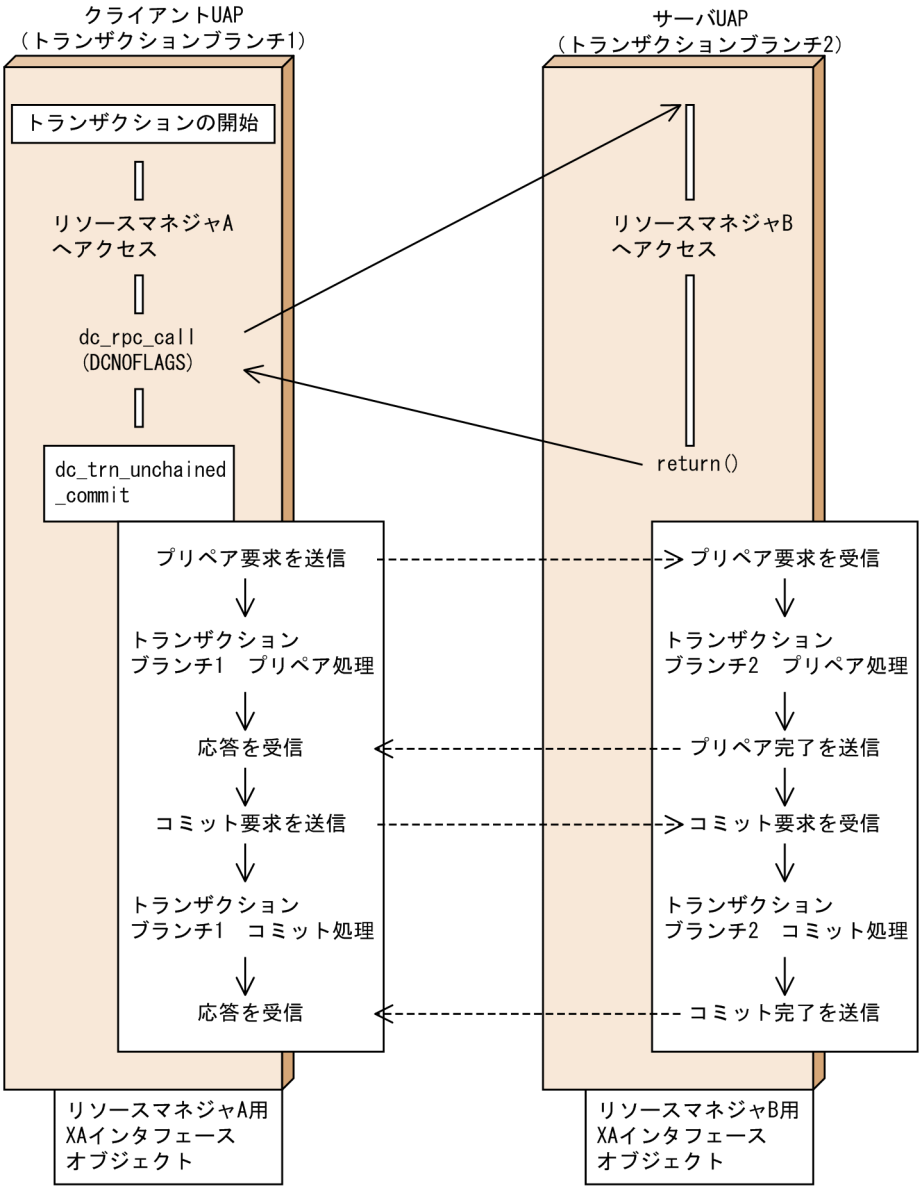
連鎖 RPC を使うと，グローバルトランザクション内のトランザクションブランチの数が少なくなるので，トランザクションをより効率的に実行できます。

(1) 通常のトランザクション処理（2 相コミット）

OpenTP1 では，X/Open の **XA インタフェース**によってトランザクションを制御しています。XA インタフェースでは，**プリペア処理**と**コミット処理**に分けて，トランザクションの同期点を取得しています。このような同期点処理を，**2 相コミット**といいます。したがって，クライアント UAP とサーバ UAP の間では，同期点処理要求の送信と確認の応答で，合計 **4 回**通信することになります。2 相コミットの処理では，トランザクション処理のプロセスは同期点を取得するまで，次のサービス要求を受け取れません。

通常のトランザクション処理（2 相コミット）の概要を，次の図に示します。

図 2-37　通常のトランザクション処理（2 相コミット）の概要



(2) コミット最適化

コミット最適化の条件を満たすと，サーバ側のトランザクションブランチで実行する同期点処理の 2 相目（コミット処理／ロールバック処理）を，クライアント側のトランザクションブランチで実行します。このことで，2 回分のプロセス間通信が不要になって，トランザクション処理の性能が上がります。

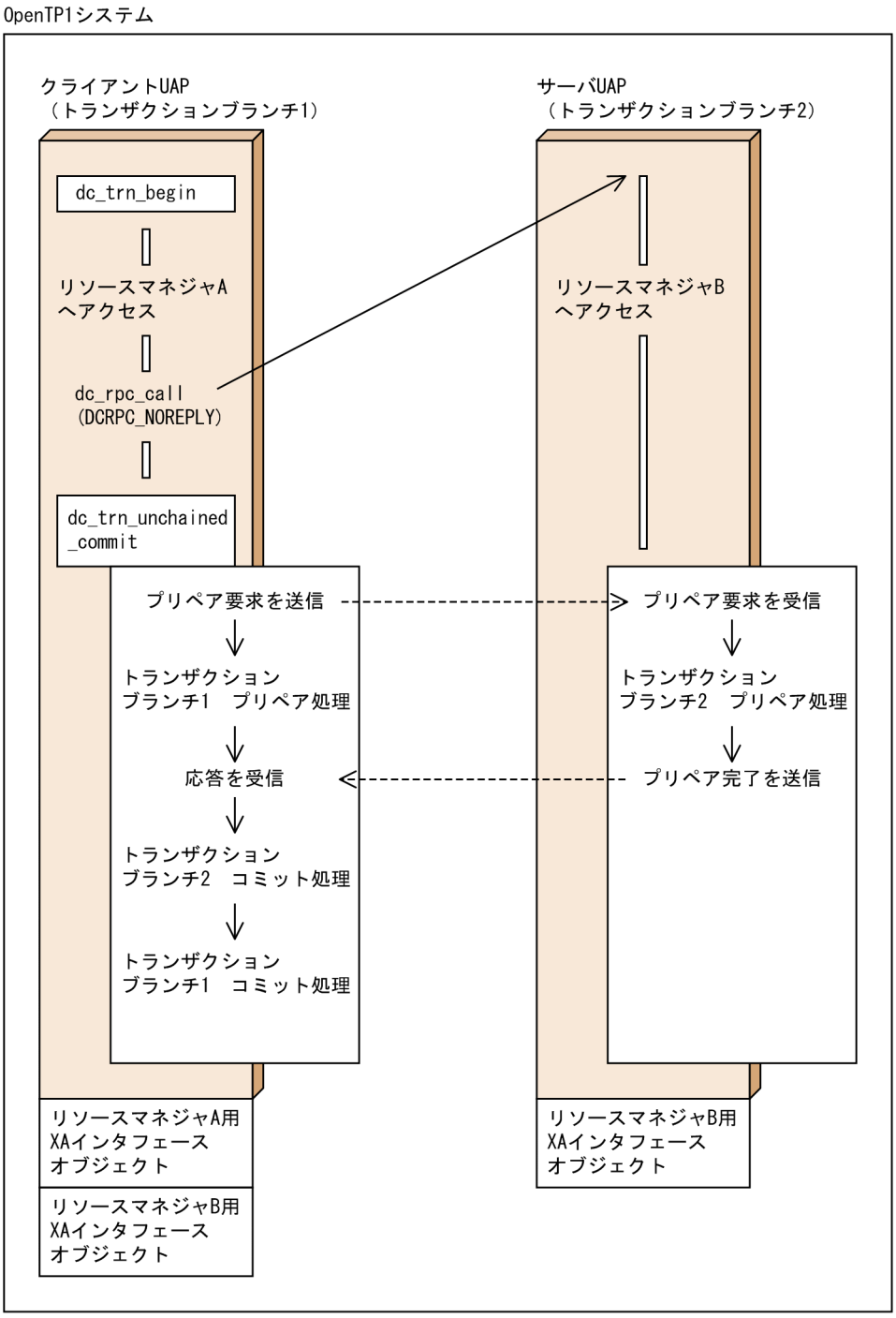
次に示す条件をすべて満たした場合に，コミット最適化は無条件に実行されます。

1. クライアント側のトランザクションブランチと，サーバ側のトランザクションブランチが，同じ OpenTP1 システム内にある。
2. サーバ側のトランザクションブランチでアクセスしたリソースマネジャの XA インタフェースオブジェクトファイルが，クライアント側のトランザクションブランチにもリンケージされている。

コミット最適化をした場合，サーバ側のトランザクションブランチは，同期点処理の 1 相目が終了した時点で，2 相目の完了を待たないで，次のサービス要求を受け付けられます。

コミット最適化の概要を次の図に示します。

図 2-38　コミット最適化の概要



(3)　プリペア最適化

プリペア最適化の条件を満たすと，サーバ側のトランザクションブランチで実行する同期点処理の 1 相目（プリペア処理）を，クライアント側のトランザクションブランチで実行します。このことで，2 回分のプロセス間通信が不要になって，トランザクション処理の性能が上がります。

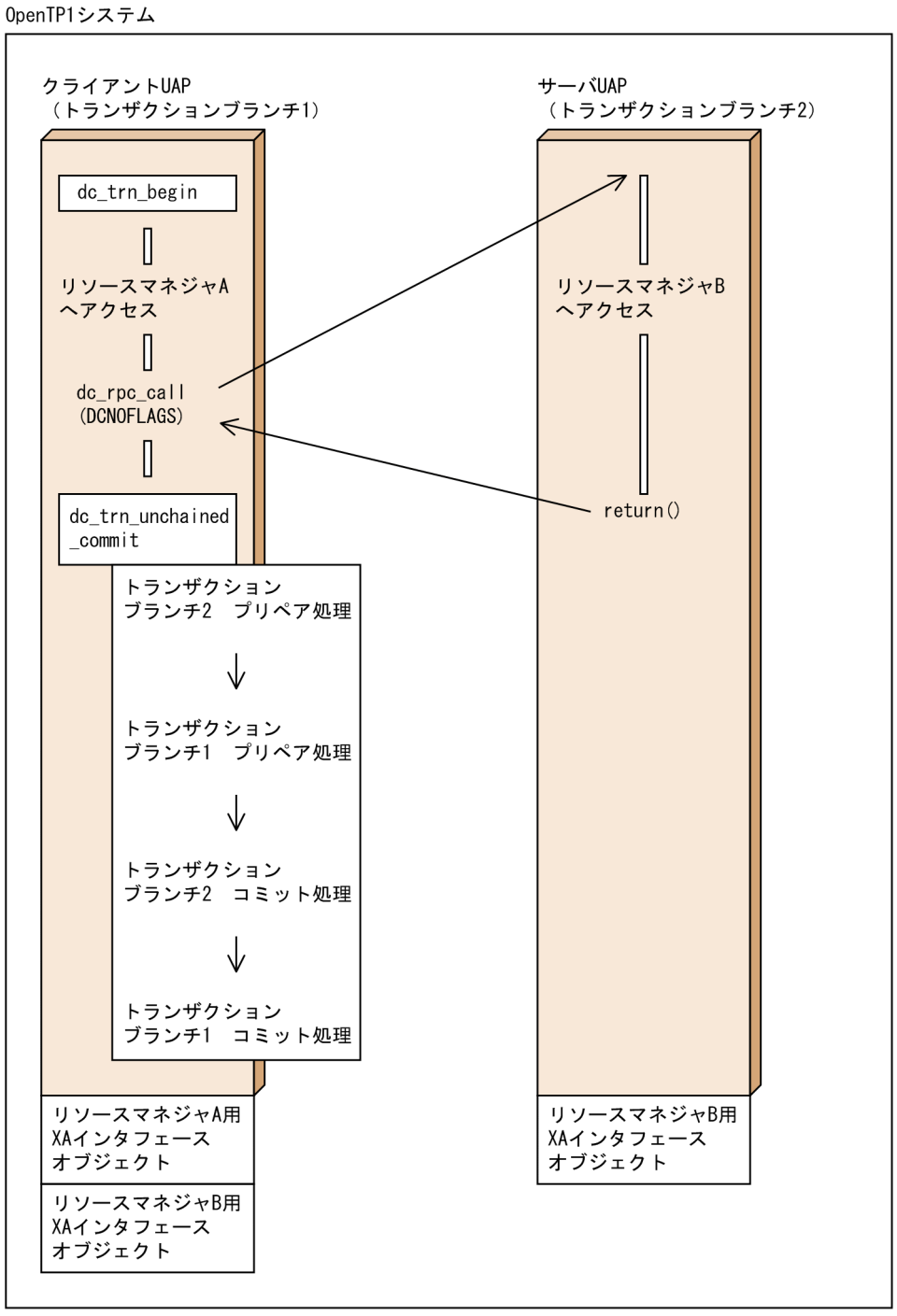
次に示す条件をすべて満たした場合に，プリペア最適化は無条件に実行されます。

1. クライアント側のトランザクションブランチと，サーバ側のトランザクションブランチが，同じOpenTP1 システム内にある。
2. サーバ側のトランザクションブランチでアクセスしたリソースマネジャの XA インタフェースオブジェクトファイルが，クライアント側のトランザクションブランチにもリンケージされている。
3. クライアント側のトランザクションブランチは，同期応答型 RPC（dc_rpc_call 関数の flags に DCNOFLAGS を設定）を使っている。

プリペア最適化をした場合は，コミット最適化も実行できるため，4 回のプロセス間通信を削減できます。また，サーバ側のトランザクションブランチでは，同期点処理の完了を待たないで，次のサービス要求を受け付けられます。

プリペア最適化の概要を次の図に示します。

図 2-39　プリペア最適化の概要



(4)　非同期プリペア最適化

非同期プリペア最適化の条件を満たすと，サーバ側のトランザクションブランチがサービス処理が終了した時点で，クライアント側のトランザクションブランチへリターンする前に，プリペア処理を実行します。このことで，2 回分のプロセス間通信が不要になって，トランザクション処理の性能が上がります。

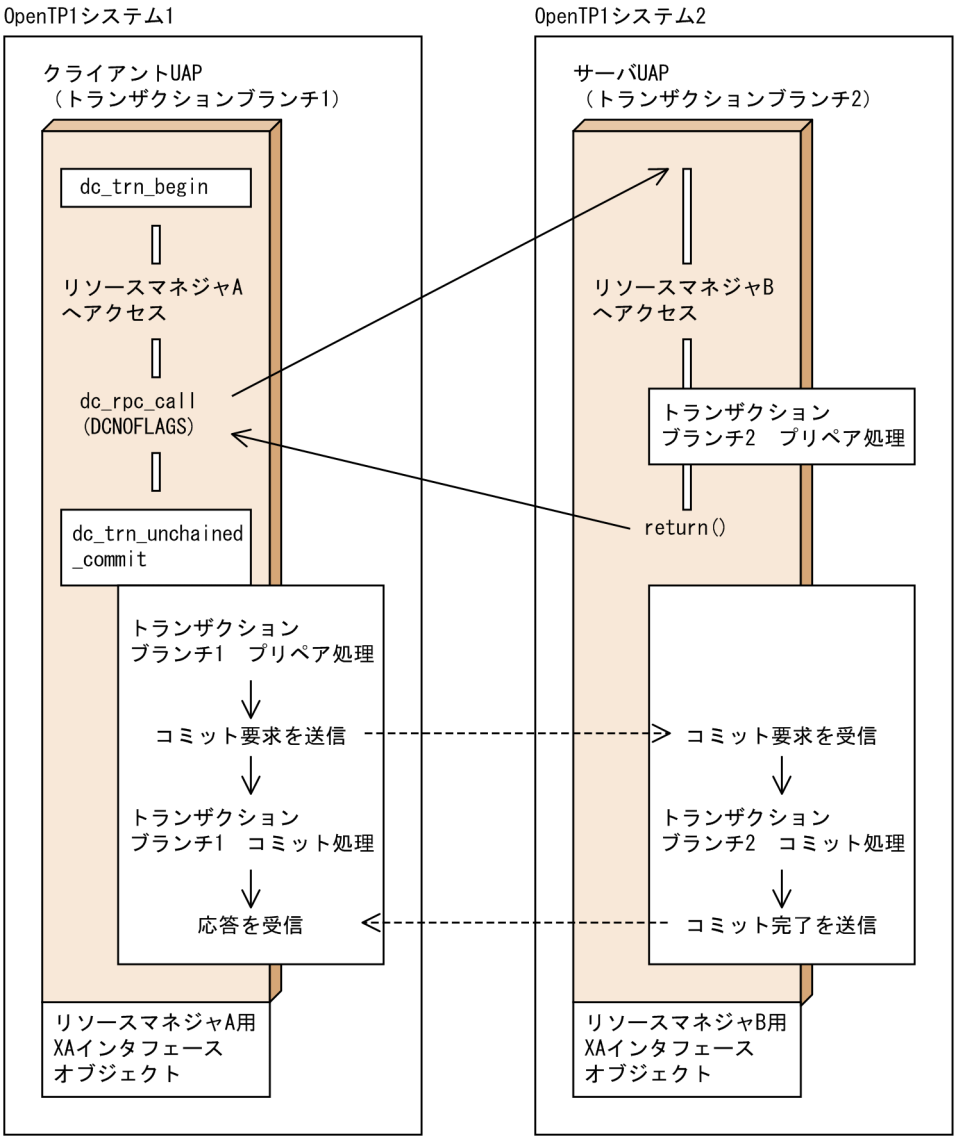
次に示す条件をすべて満たした場合に，非同期プリペア最適化は実行されます。

1. クライアント側の UAP で，ユーザサービス定義の trn_optimum_item オペランドに asyncprepare を指定している。
2. プリペア最適化が実行できない条件であること（実行できる場合は，プリペア最適化が非同期プリペアよりも優先される）。
3. クライアント側のトランザクションブランチは，同期応答型 RPC（dc_rpc_call 関数の flags に DCNOFLAGS を設定）を使っている。

非同期プリペア最適化をした場合，RPC の応答時間が通常のトランザクション処理よりも長くなります。また，クライアント側のトランザクションブランチの OpenTP1 が，トランザクション処理中に異常終了した場合，ジャーナル取得の関係で，サーバ側のトランザクションブランチの OpenTP1 も異常終了する場合があります。

非同期プリペア最適化の概要を次の図に示します。

図 2-40　非同期プリペア最適化の概要



(5) 1 相最適化

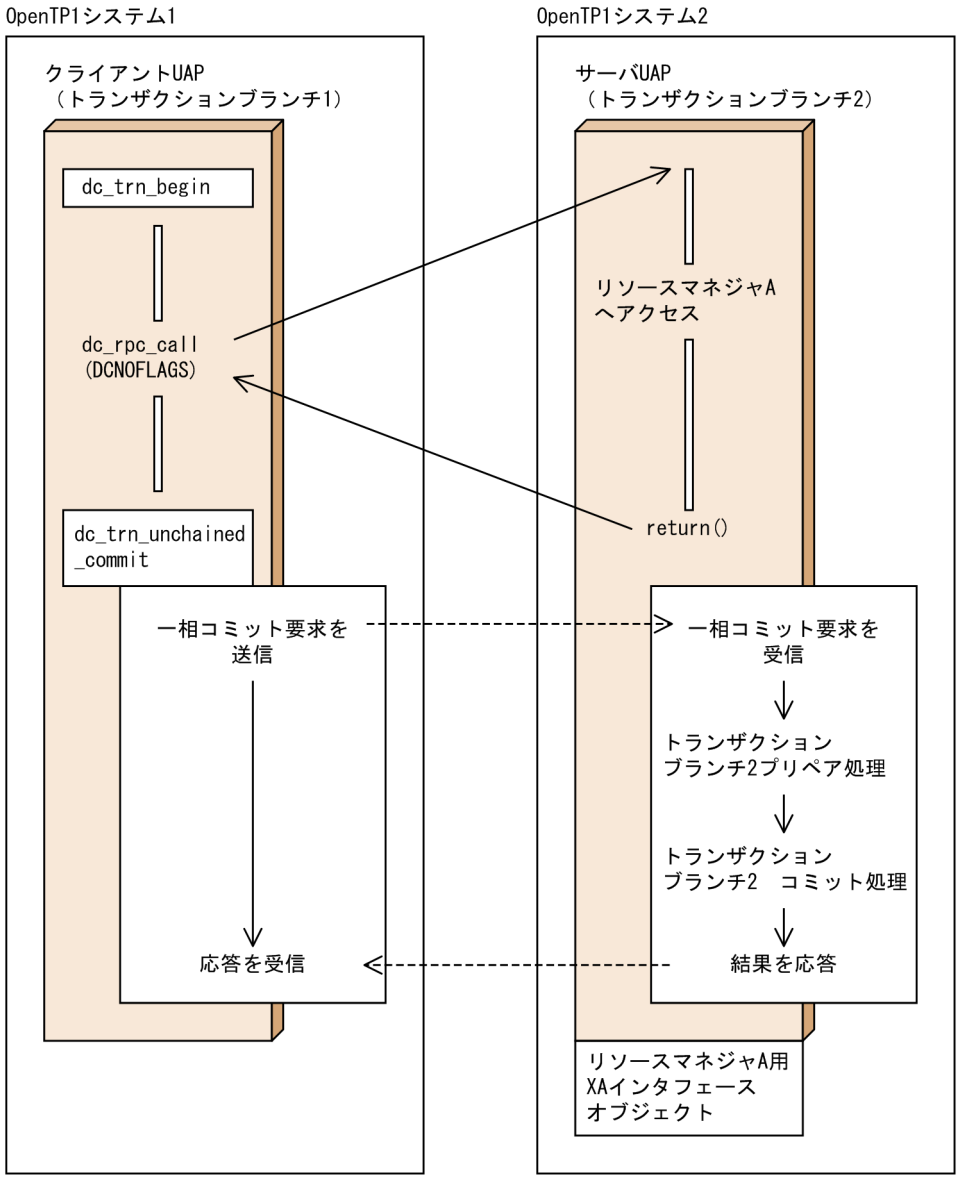
1 相最適化の条件を満たすと，クライアント側のトランザクションブランチがリソースマネジャへアクセスしないため，サーバ側のトランザクションブランチの同期点処理だけで済みます。このことで，2 回分のプロセス間通信が不要になって，トランザクション処理の性能が上がります。

次に示す条件をすべて満たした場合に，1 相最適化は実行されます。

1. クライアント側のトランザクションブランチにリンケージされているリソースマネジャは，動的登録をするリソースマネジャだけである。
2. クライアント側のトランザクションブランチは，リソースマネジャへのアクセスやユーザジャーナルの出力をしていない。
3. クライアント側のトランザクションブランチには，子トランザクションブランチが一つだけである。

1 相最適化の概要を次の図に示します。

図 2-41　1 相最適化の概要



(6)　リードオンリー最適化

リードオンリー最適化の条件を満たすと，サーバ側のトランザクションブランチが更新処理をしていない場合に，同期点処理の 2 相目を実行しません。このことで，2 回分のプロセス間通信が不要になって，トランザクション処理の性能が上がります。

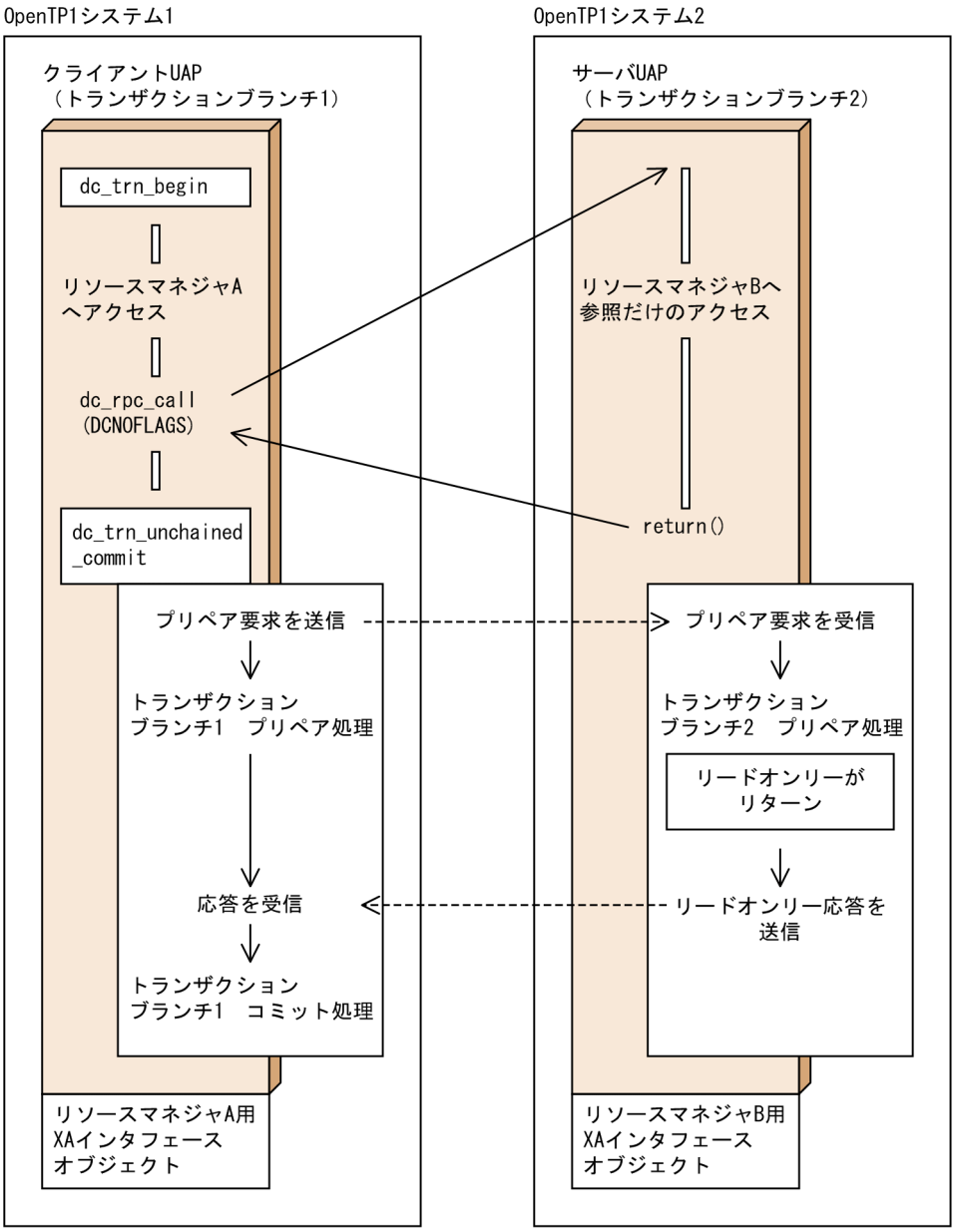
次に示す条件をすべて満たした場合に，リードオンリー最適化は実行されます。

1. サーバ側のトランザクションブランチは，リソースの更新処理（参照処理は除く），およびユーザジャーナルの出力をしていない。
2. クライアント側のトランザクションブランチには，子トランザクションブランチが一つだけである。

リードオンリー最適化をした場合，サーバ側のトランザクションブランチは，同期点処理の 1 相目が終了した時点で，2 相目の完了を待たないで，次のサービス要求を受け付けられます。

リードオンリー最適化の概要を次の図に示します。

図 2-42　リードオンリー最適化の概要



(7)　ノーアクセス最適化

ノーアクセス最適化の条件を満たすと，サーバ側のトランザクションブランチがリソースマネジャへアクセスしていない場合，同期点処理をしません。このことで，4 回分のプロセス間通信が不要になって，トランザクション処理の性能が上がります。

次に示す条件をすべて満たした場合に，ノーアクセス最適化は実行されます。

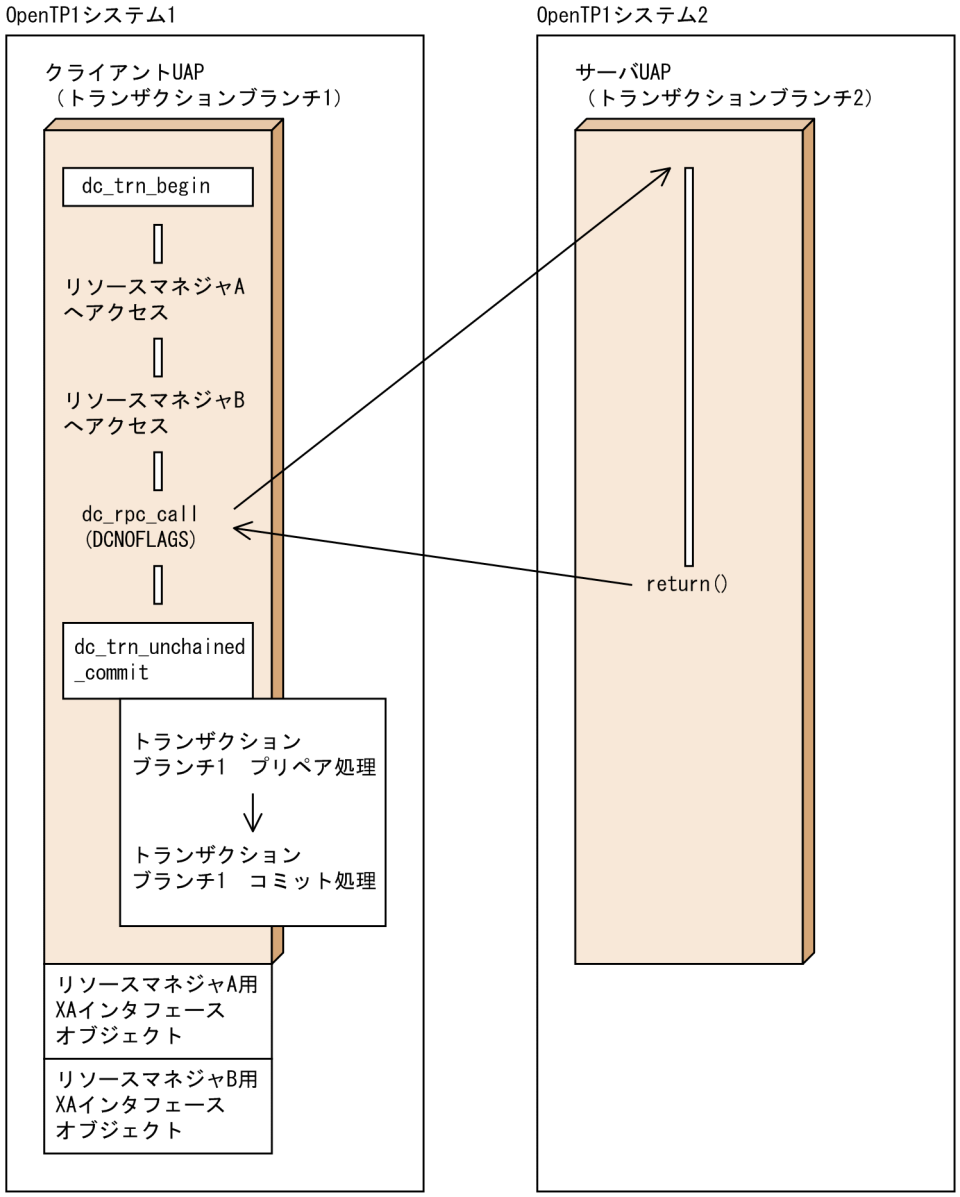
1. クライアント側のトランザクションブランチは，同期応答型 RPC（dc_rpc_call 関数の flags に DCNOFLAGS を設定）を使っている。

2. サーバ側のトランザクションブランチにリンケージされているリソースマネジャは，動的登録をするリソースマネジャだけである。
3. サーバ側のトランザクションブランチは，リソースマネジャへのアクセス，およびユーザジャーナルへの出力をしていない。
4. サーバ側のトランザクションブランチには，子トランザクションブランチがない。または子トランザクションブランチが存在しても，その子トランザクションブランチはリードオンリー最適化ができる状態である。

ノーアクセス最適化をした場合，サーバ側のトランザクションブランチは，同期点処理の完了を待たないで，次のサービス要求を受け付けられます。

ノーアクセス最適化の概要を次の図に示します。

図 2-43　ノーアクセス最適化の概要



(8) ロールバック最適化

ロールバック最適化の条件を満たすと，サーバ側のトランザクションブランチで，ロールバックの関数を使った場合，ほかのトランザクションブランチと同期を取らないでロールバックします。また，ほかのトランザクションブランチは同期点処理の 1 相目を実行しません。このことで，2 回分のプロセス間通信が不要になって，トランザクション処理の性能が上がります。

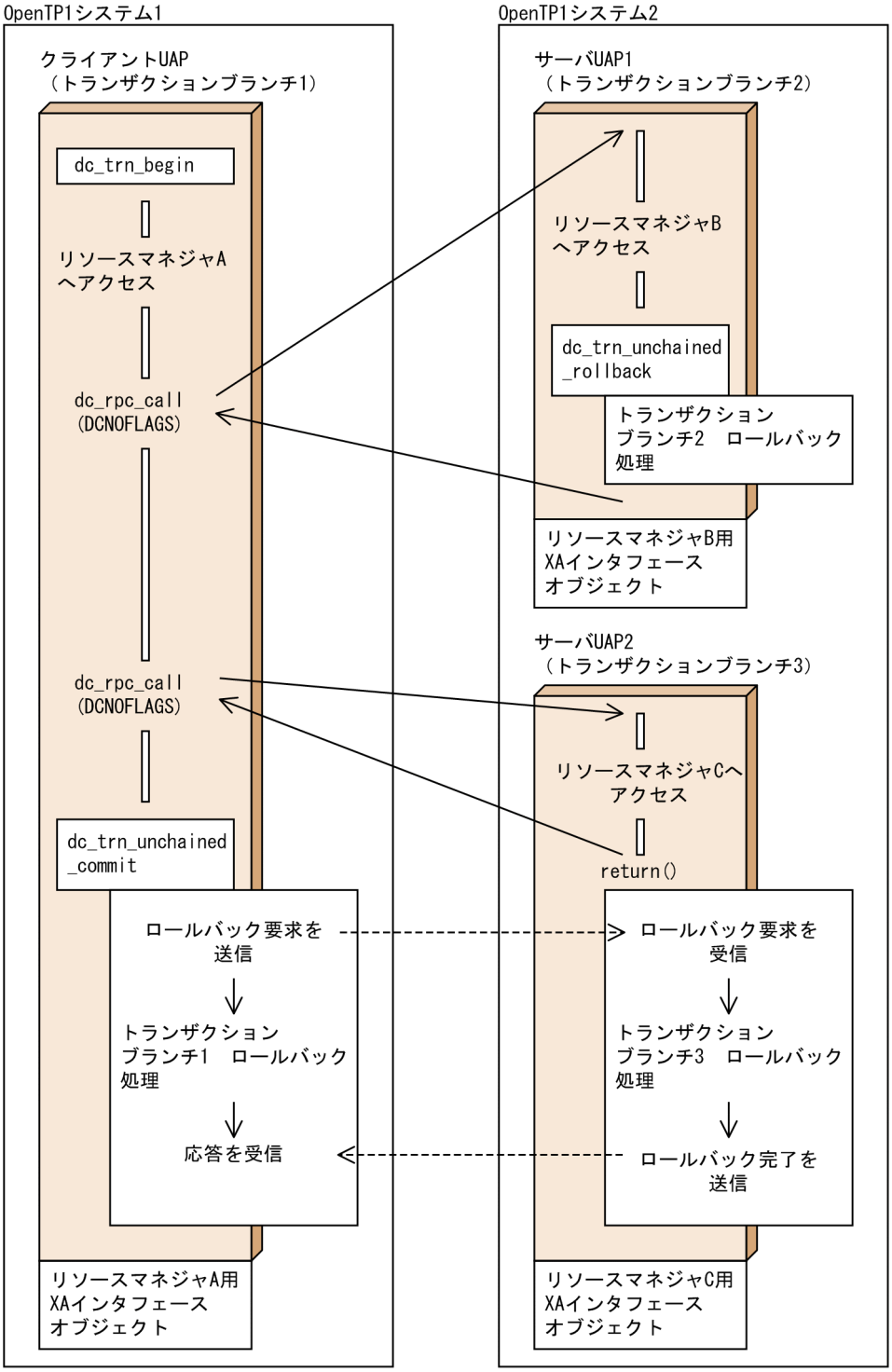
次に示す条件をすべて満たした場合に，ロールバック最適化は実行されます。

1. クライアント側のトランザクションブランチは，同期応答型 RPC（dc_rpc_call 関数の flags に DCNOFLAGS を設定）を使っている。
2. サーバ側のトランザクションブランチで，ロールバック関数を使っている。

ロールバック最適化をした場合，サーバ側のトランザクションブランチは，同期点処理の完了を待たないで，次のサービス要求を受け付けられます。

ロールバック最適化の概要を次の図に示します。

図 2-44　ロールバック最適化の概要



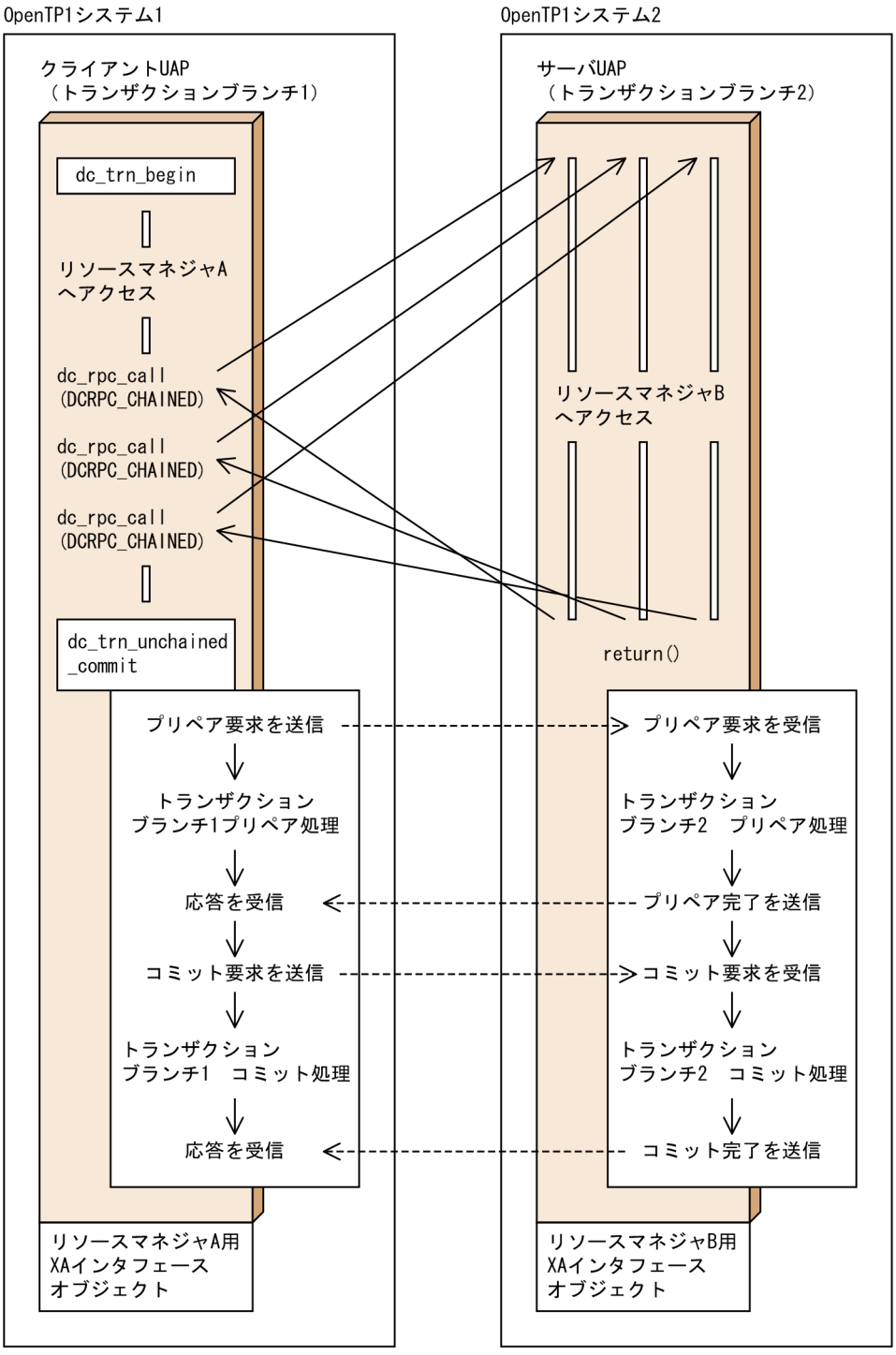
(9)　連鎖 RPC を使った最適化

通常，トランザクションブランチからトランザクション属性を指定した UAP へサービスを要求すると，サーバ側のトランザクションブランチのプロセスは，別のトランザクションブランチとして実行されます。ただし，同じサービスグループ（同じサービスでなくても良い）へ連鎖 RPC（dc_rpc_call 関数の flags に DCRPC_CHAINED を設定）を使うと，その連鎖 RPC を終了するまで一つのトランザクションブラン

チとして実行されます。連鎖 RPC の条件を満たすと，グローバルトランザクション内のトランザクションブランチの数が削減されて，トランザクション処理の性能が上がります。

連鎖 RPC を使った最適化の概要を次の図に示します。

図 2-45　連鎖 RPC を使った最適化の概要



2.3.6　現在のトランザクションに関する情報を報告

UAP から dc_trn_info 関数【CBLDCTRN('INFO ')】を呼び出すと，そのリターン値で，トランザクションとして稼働中かどうかを知ることができます。

2.3.7　ヒューリスティック発生時の処置

ノード間で通信障害が起こって，トランザクションブランチ間で連絡できなくなった場合，各ノードで**コマンド**を実行して同期点を取得する必要があります。各ノードで同期点を取得すると，グローバルトランザクションの中で，あるトランザクションブランチがコミット，あるトランザクションブランチがロールバックとなることがあります。このように，ノードごとに同期点を取得することを**ヒューリスティック決定**といいます。ヒューリスティック決定の状態のときは，UAP でグローバルトランザクションの同期点を取得しようとすると，関数がエラーリターンします。ヒューリスティック決定が原因で，関数がエラーリターンする値を次に示します。

- ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合
  DCTRNER_HEURISTIC（00903）
- 障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果が判明しない場合
  DCTRNER_HAZARD（00904）

このリターン値が戻る原因になった UAP やリソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照して確認してください。

2.3.8　トランザクション処理での注意事項

(1)　ユーザサービス定義との関連

トランザクションとして実行中のサービスから，トランザクション属性であることを指定（atomic_update オペランドに Y を指定）したサービスを要求する場合は，次のことに気を付けてください。

1. サーバ UAP のユーザサービス定義の最大プロセス数（parallel_count オペランド）は余裕を持った値を指定しておいてください。サーバ UAP の処理が終了しても，グローバルトランザクションの同期点処理が完了するまで，ほかのクライアント UAP にはサービスを提供しません（ただし，同期点処理の最適化ができる場合を除きます）。この場合，トランザクションが長時間続くと，その間にサービスを要求した，異なるクライアント UAP の数だけのプロセスを占有することになります。そのため，トランザクションの性能が低下するおそれがあります。
2. ユーザサービス定義のサービス要求滞留値（balance_count オペランド）に指定した値によっては，マルチサーバ機能を使ったユーザサーバでも非常駐プロセスが増えないで，RPC がタイムアウトになる場合があります。balance_count オペランドには，サーバ UAP の負荷に応じた最適な値を指定してください。

次の場合には，必ず balance_count オペランドに 0 を指定してください。

- 非常駐プロセスだけで構成されるユーザサーバでリカーシブコールをする場合（例えば，parallel_count=0,2 の場合）。
- 一つの常駐プロセスと，ほかの非常駐プロセスで構成されるユーザサーバでリカーシブコールをする場合（例えば，parallel_count=1,2 の場合）。

(2) トランザクション処理の時間監視について

トランザクションの開始から同期点処理までの**時間監視**では，トランザクション内で呼び出した dc_rpc_call 関数がリターンするまでの時間を含めるか含めないかを選択できます。この指定はユーザサービス定義，ユーザサービスデフォルト定義，トランザクションサービス定義の trn_expiration_time_suspend オペランドで指定します。

trn_expiration_time_suspend オペランドに指定する値とトランザクションの時間監視の詳細については，マニュアル「OpenTP1 システム定義」を参照してください。

2.4　システム運用の管理

ここでは，UAP から関数を呼び出して OpenTP1 のコマンドを実行する方法，SUP の開始処理完了の報告，および UAP から関数を呼び出して UAP の状態を取得する方法について説明します。

2.4.1　運用コマンドの実行

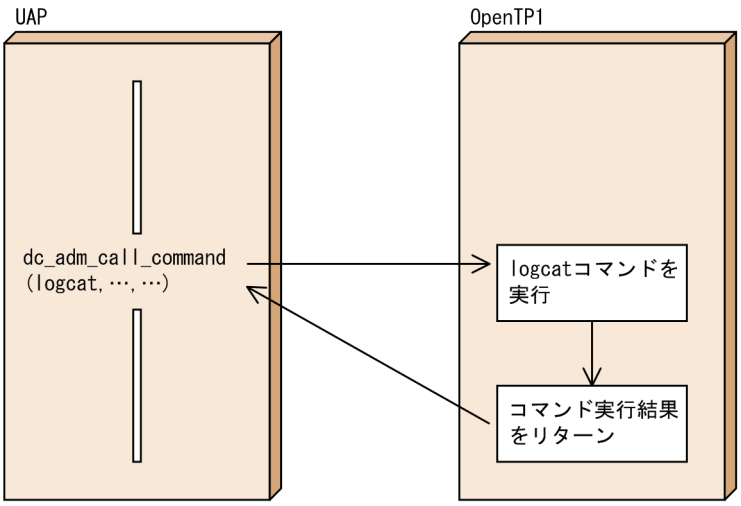
OpenTP1 システムの運用を支援するために，オンライン中に入力できる OpenTP1 のコマンドを，UAP から dc_adm_call_command 関数【CBLDCADM('COMMAND ')】で実行できます。コマンドの実行結果は，UAP にリターンします。リターンする内容は**標準出力**，または**標準エラー出力**に出力される値です。

コマンドを実行する UAP には，次に示す指定をして，コマンドがあるディレクトリをコマンドのサーチパスに定義しておいてください。

- TP1/Server Base の場合：
  ユーザサービス定義の putenv PATH に環境変数を指定
- TP1/LiNK の場合：
  TP1/LiNK の環境設定時にサーチパスを追加

dc_adm_call_command 関数を使った OpenTP1 のコマンド実行の概要を次の図に示します。

図 2-46　dc_adm_call_command 関数を使った OpenTP1 のコマンド実行の概要



(1)　dc_adm_call_command 関数で実行できる OpenTP1 のコマンド

OpenTP1 のコマンドと UAP からの実行可否を次の表に示します。

表 2-1　OpenTP1 のコマンドと UAP からの実行可否

機能		コマンド名	UAP から実行
システム管理	OpenTP1 の OS への登録と削除	dcsetup	×
	プロセスサービスの再起動および定義の反映	dcreset	×
	OpenTP1 の内部制御用資源の確保と解放	dcmakeup	×
	OpenTP1 の開始	dcstart	×
	OpenTP1 の終了	dcstop	○※1
	システム統計情報の取得開始，終了	dcstats	○
	マルチノードエリア，サブエリアの開始	dcmstart	○
	マルチノードエリア，サブエリアの終了	dcmstop	○
	シナリオテンプレートからの OpenTP1 コマンドの実行	dcjcmdex	×
	システム定義のオペランドの指定	dcjchconf	×
	ドメイン定義ファイルの更新	dcjnamch	○
	OpenTP1 ノードの状態表示	dcndls	○
	共用メモリの状態表示	dcshmls	○
	一時クローズ処理の実行状態の表示	rpcstat	○
	標準出力，標準エラー出力のリダイレクト	prctee	×
	prctee プロセスの停止と再開始	prctctrl	×
	保守資料の取得	dcrasget	○
	システム統計情報の標準出力へのリアルタイム編集出力	dcreport	○
	トラブルシュート情報の削除	dccspool	○
	システム定義のチェック	dcdefchk	×
	製品情報の表示	dcpplist	○
リモート API 管理	rap リスナーおよび rap サーバの状態表示	rapls	×
	リモート API 機能の実行環境の設定	rapsetup	×
	リモート API 機能に使用する定義の自動生成	rapdfgen	×
サーバ管理	サーバの開始	dcsvstart	○
	サーバの終了	dcsvstop	○
	サーバの状態表示	prcls	○
	ユーザサーバ，およびユーザサーバから起動されるコマンドのサーチパスの表示	prcpathls	○

機能		コマンド名	UAP から実行
サーバ管理	ユーザサーバ，およびユーザサーバから起動されるコマンドのサーチパスの変更	prcpath	○※2
	UAP 共用ライブラリのサーチパスの表示	prcdlpathls	○
	UAP 共用ライブラリのサーチパスの変更	prcdlpath	○※2
	OpenTP1 のプロセスの強制停止	prckill	○
スケジュール管理	スケジュールの状態表示	scdls	○
	スケジュールの閉塞	scdhold	○
	スケジュールの再開始	scdrles	○
	プロセス数の変更	scdchprc	○
	プロセスの停止および再起動	scdrsprc	○
トランザクション管理	トランザクションの状態表示	trnls	○
	トランザクションの強制コミット	trncmt	○
	トランザクションの強制ロールバック	trnrbk	○
	トランザクションの強制終了	trnfgt	○
	トランザクション統計情報の取得開始，終了	trnstics	○
	未決着トランザクション情報ファイルの削除	trndlinf	○
	OSI TP 通信の未決着トランザクション情報の表示	tptrnls	○
XA リソース管理	XAR イベントトレース情報の表示	xarevtr	×
	XAR ファイルの状態表示	xarfills	○
	XAR トランザクション状態の変更	xarforce	○
	XA リソースサービスの閉塞	xarhold	○
	XAR ファイルの作成	xarinit	×
	XAR トランザクション情報の表示	xarls	○
	XA リソースサービスの閉塞解除	xarrles	○
	XAR ファイルの削除	xarrm	×
排他管理	排他情報の表示	lckls	○
	排他制御用テーブルのプール情報の表示	lckpool	○
	デッドロック情報ファイルとタイムアウト情報ファイルの削除	lckrminf	○
ネーム管理	OpenTP1 起動確認，キャッシュ削除	namalivechk	○
	ドメイン代表スケジュールサービスの登録，削除	namdomainsetup	○

機能		コマンド名	UAP から実行
ネーム管理	ドメイン構成の変更（システム共通定義使用）	namndchg	○
	ドメイン構成の変更（ドメイン定義ファイル使用）	namchgfl	○
	起動通知情報の強制的無効化	namunavl	×
	OpenTP1 のサーバ情報の表示	namsvinf	○
	RPC 抑止リストの操作	namblad	○
	ノードリスト情報の削除（ノード自動追加機能使用時）	namndrm	×
	マネジャノードの変更（ノード自動追加機能使用時）	nammstr	×
	ノードリストファイルの作成（ノード自動追加機能使用時）	namnlcre	×
	ノードリストファイルの内容表示（ノード自動追加機能使用時）	namnldsp	○
	ノードリストファイルの削除（ノード自動追加機能使用時）	namnldel	×
	ノードのオプション情報の変更（ノード自動追加機能使用時）	namndopt	○
メッセージログ管理	メッセージログファイルの内容表示	logcat	○
	メッセージログのリアルタイム出力機能の切り替え	logcon	○
監査ログ	監査ログ機能の環境設定	dcauditsetup	×
OpenTP1 ファイル管理	OpenTP1 ファイルシステムの初期設定	filmkfs	×
	OpenTP1 ファイルシステムの状態表示	filstatfs	○
	OpenTP1 ファイルシステムの内容表示	fills	○
	OpenTP1 ファイルシステムのバックアップ	filbkup	×
	OpenTP1 ファイルシステムのリストア	filrstr	×
	OpenTP1 ファイルグループの変更	filchgrp	○
	OpenTP1 ファイルのアクセス許可モードの変更	filchmod	○
	OpenTP1 ファイル所有者の変更	filchown	○
ステータスファイル管理	ステータスファイルの作成，初期設定	stsinit	×
	ステータスファイルの状態表示	stsls	○
	ステータスファイルの内容表示	stsfills	○
	ステータスファイルのオープン	stsopen	○
	ステータスファイルのクローズ	stsclose	○
	ステータスファイルの削除	stsrm	○
	ステータスファイルのスワップ	stsswap	○

機能		コマンド名	UAP から実行
ジャーナル関係のファイル管理	ジャーナル関係のファイルの初期設定	jnlinit	×
	ジャーナル関係のファイル情報の表示	jnlls	○
	再開始中読み込み済ジャーナル関係のファイル情報の表示	jnlrinf	×
	ジャーナル関係のファイルのオープン	jnlopnfg	○
	ジャーナル関係のファイルのクローズ	jnlclsfg	○
	ジャーナル関係の物理ファイルの割り当て	jnladdpf	○
	ジャーナル関係の物理ファイルの削除	jnldelpf	○
	ジャーナル関係のファイルのスワップ	jnlswpfg	○
	ジャーナル関係のファイルの削除	jnlrm	×
	ジャーナル関係のファイルのステータス変更	jnlchgfg	×
	ジャーナル関係のファイルのアンロード	jnlunlfg	×
	自動アンロード機能の制御	jnlatunl	×
	ジャーナル関係のファイルの回復	jnlmkrf	×
	ファイル回復用ジャーナルの集積	jnlcolc	×
	アンロードジャーナルファイルの複写	jnlcopy	×
	アーカイブ状態の表示	jnlarls	○
	アンロードジャーナルファイル，またはグローバルアーカイブアンロードジャーナルファイルの編集出力	jnledit	×
	アンロードジャーナルファイル，またはグローバルアーカイブアンロードジャーナルファイルのレコード出力	jnlrput	×
	アンロードジャーナルファイル，およびグローバルアーカイブアンロードジャーナルファイルの時系列ソート，およびマージ	jnlsort	×
	稼働統計情報の出力	jnlstts	×
	MCF 稼働統計情報の出力	jnlmcst	×
	リソースグループの接続の強制解除	jnlardis	×
DAM ファイル管理	物理ファイルの初期設定	damload	×
	論理ファイルの状態表示	damls	○
	論理ファイルの追加	damadd	○
	論理ファイルの切り離し	damrm	○
	論理ファイルの論理閉塞	damhold	○
	論理ファイルの閉塞解除	damrles	○

機能		コマンド名	UAP から実行
DAM ファイル管理	物理ファイルの削除	damdel	×
	物理ファイルのバックアップ	dambkup	×
	物理ファイルのリストア	damrstr	×
	論理ファイルの回復	damfrc	×
	キャッシュブロック数のしきい値の設定	damchdef	○
	キャッシュブロック数の取得	damchinf	○
TAM ファイル管理	TAM ファイルの初期作成	tamcre	×
	TAM テーブルの状態表示	tamls	○
	TAM テーブルの追加	tamadd	○
	TAM テーブルの切り離し	tamrm	○
	TAM テーブルの論理閉塞	tamhold	○
	TAM テーブルの閉塞解除	tamrles	○
	TAM テーブルのロード	tamload	○
	TAM テーブルのアンロード	tamunload	○
	TAM ファイルの削除	tamdel	×
	TAM ファイルのバックアップ	tambkup	×
	TAM ファイルのリストア	tamrstr	×
	TAM ファイルの回復	tamfrc	×
	TAM 排他資源名称の変換	tamlckls	○
	ハッシュ形式の TAM ファイルおよび TAM テーブルのシノニム情報の表示	tamhsls	×
メッセージキューファイル管理	キューグループの状態表示	quels	○
	メッセージキュー用物理ファイルの割り当て	queinit	×
	メッセージキュー用物理ファイルの削除	querm	×
リソースマネジャ管理	リソースマネジャの情報の表示	trnlsrm	×
	リソースマネジャの登録	trnlnkrm	×
	トランザクション制御用オブジェクトファイルの作成	trnmkobj	×
トレース管理	UAP トレースの編集出力	uatdump	×
	RPC トレースのマージ	rpcmrg	×
	RPC トレースの出力	rpcdump	×

機能		コマンド名	UAP から実行
トレース管理	共用メモリダンプの出力	usmdump	○
性能検証用トレース管理	トレース情報ファイルの編集出力	prfed	×
	トレース情報ファイルの取り出し	prfget	×
リアルタイム統計情報サービス管理	RTS ログファイルの編集出力	rtsedit	×
	リアルタイム統計情報の標準出力への出力	rtsls	×
	リアルタイム統計情報サービスの実行環境の設定	rtssetup	×
	リアルタイム統計情報の設定変更	rtsstats	×
OpenTP1 解析支援	性能検証用トレース解析	dcalzprf	×
コネクション管理	コネクションの状態表示	mcftlscn	○
	コネクションの確立	mcftactcn	○
	コネクションの解放	mcftdctcn	○
	コネクションの切り替え	mcftchcn	○
	ネットワークの状態表示	mcftlsln	○
	サーバ型コネクションの確立要求の受付開始	mcftonln	○
	サーバ型コネクションの確立要求の受付終了	mcftofln	○
	メッセージ多重処理状況の表示	mcftlstrd	○
アプリケーション管理	アプリケーションの状態表示	mcfalsap	○
	アプリケーションの閉塞	mcfadctap	○
	アプリケーションの閉塞解除	mcfaactap	○
	アプリケーション異常終了回数の初期化	mcfaclcap	○
	アプリケーション起動要求の状態表示	mcfalstap	○
	アプリケーションに関するタイマ起動要求の削除	mcfadltap	○
アプリケーション運用支援	アプリケーションプログラムの起動	mcfuevt	○
論理端末管理	論理端末の状態表示	mcftlsle	○
	論理端末の閉塞	mcftdctle	○
	論理端末の閉塞解除	mcftactle	○
	論理端末のメッセージキューの先頭スキップ	mcftspqle	○
	論理端末の出力キュー処理の保留	mcfthldoq	○
	論理端末の出力キュー処理の保留解除	mcftrlsoq	○

機能		コマンド名	UAP から実行
論理端末管理	論理端末の出力キュー削除	mcftdlqle	○
	論理端末に関するメッセージジャーナルの取得開始	mcftactmj	○
	論理端末に関するメッセージジャーナルの取得終了	mcftdctmj	○
	論理端末に対する継続問い合わせ応答処理の強制終了	mcftendct	○
	端末代行の開始	mcftstalt	○
	端末代行の終了	mcftedalt	○
サービスグループ管理	サービスグループの状態表示	mcftlssg	○
	サービスグループの閉塞	mcftdctsg	○
	サービスグループの閉塞解除	mcftactsg	○
	サービスグループの入力キュー処理の保留	mcfthldiq	○
	サービスグループの入力キュー処理の保留解除	mcftrlsiq	○
	サービスグループの入力キュー削除	mcftdlqsg	○
サービス管理	サービスの状態表示	mcftlssv	○
	サービスの閉塞	mcftdctsv	○
	サービスの閉塞解除	mcftactsv	○
バッファ管理	バッファグループの使用状況表示	mcftlsbuf	○
マップ管理	マップファイルのパス名変更	dcmapchg	×
	マップファイルのロード済み資源の表示	dcmapls	×
キュー管理	入出力キューの内容複写	mcftdmpqu	○
MCF トレース取得管理	MCF トレースファイルの強制スワップ	mcftswptr	○
	MCF トレース取得の開始	mcftstrtr	○
	MCF トレース取得の終了	mcftstptr	○
MCF 稼働統計情報管理	MCF 稼働統計情報の編集	mcfreport	×
	MCF 稼働統計情報の出力	mcfstats	○
MCF 通信サービス管理	MCF 通信サービスの部分停止	mcftstop	×
	MCF 通信サービスの部分開始	mcftstart	×
	MCF 通信サービスの状態参照と開始待ち合わせ	mcftlscom	×
ユーザタイマ監視	ユーザタイマ監視の状態表示	mcftlsutm	○

（凡例）

○：UAP の処理から実行できます。

×：UAP の処理から実行できません。

注※1

dcstop コマンドを UAP から実行する場合は，バックグラウンドで実行してください。

注※2

コマンドで変更された環境は，呼び出し元の UAP では有効になりません。

2.4.2　ユーザサーバの開始処理完了の報告

ユーザサーバの開始処理が完了したことを報告する dc_adm_complete 関数【CBLDCADM('COMPLETE')】は，SUP で必ず呼び出します。dc_rpc_open 関数を呼び出したあとで，開始処理の完了を OpenTP1 に連絡するために，dc_adm_complete 関数を必ず呼び出してください。

SPP と MHP では，dc_rpc_mainloop 関数，または dc_mcf_mainloop 関数が正常に実行されたことで開始処理の完了と見なすので，dc_adm_complete 関数を呼び出す必要はありません。

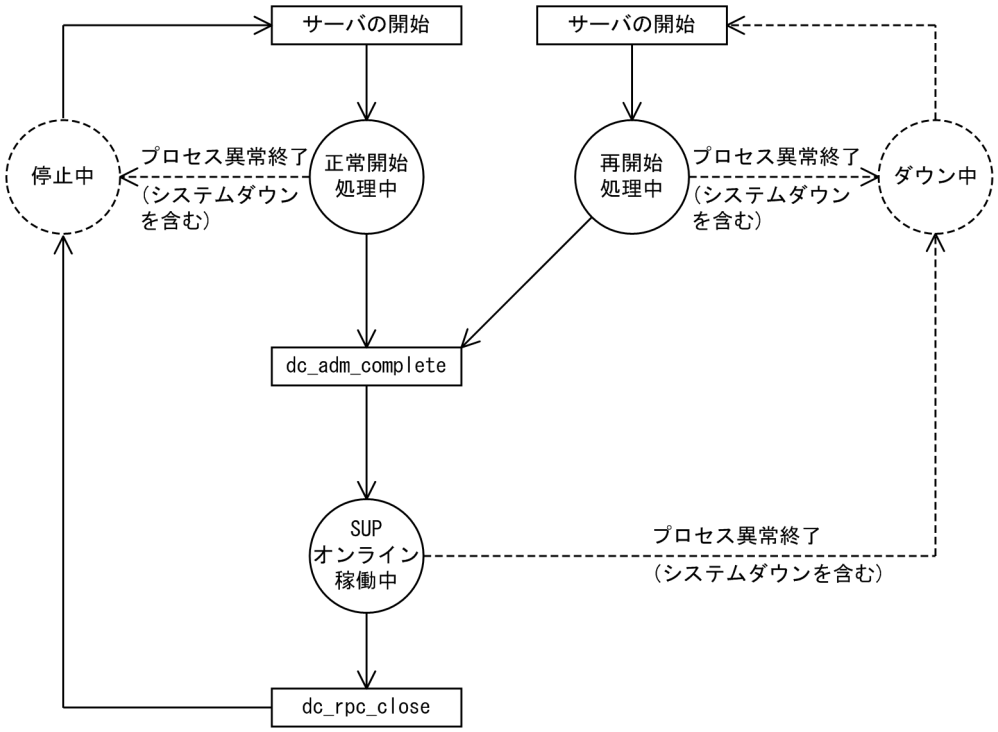
オフラインの業務をする UAP からは，dc_adm_complete 関数を呼び出せません。

2.4.3　ユーザサーバの状態の検知

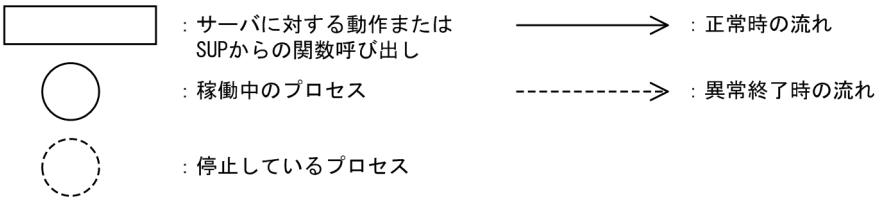
ユーザサーバが起動中かどうかなど，ユーザサーバの状態を UAP で知ることができます。UAP から dc_adm_status 関数【CBLDCADM('STATUS ')】を呼び出すと，OpenTP1 からユーザサーバの状態が返されます。

ユーザサーバの状態遷移を以降の図に示します。図に示すサーバの状態が OpenTP1 から報告されます。

図 2-47　ユーザサーバの状態遷移（SUP）



（凡例）

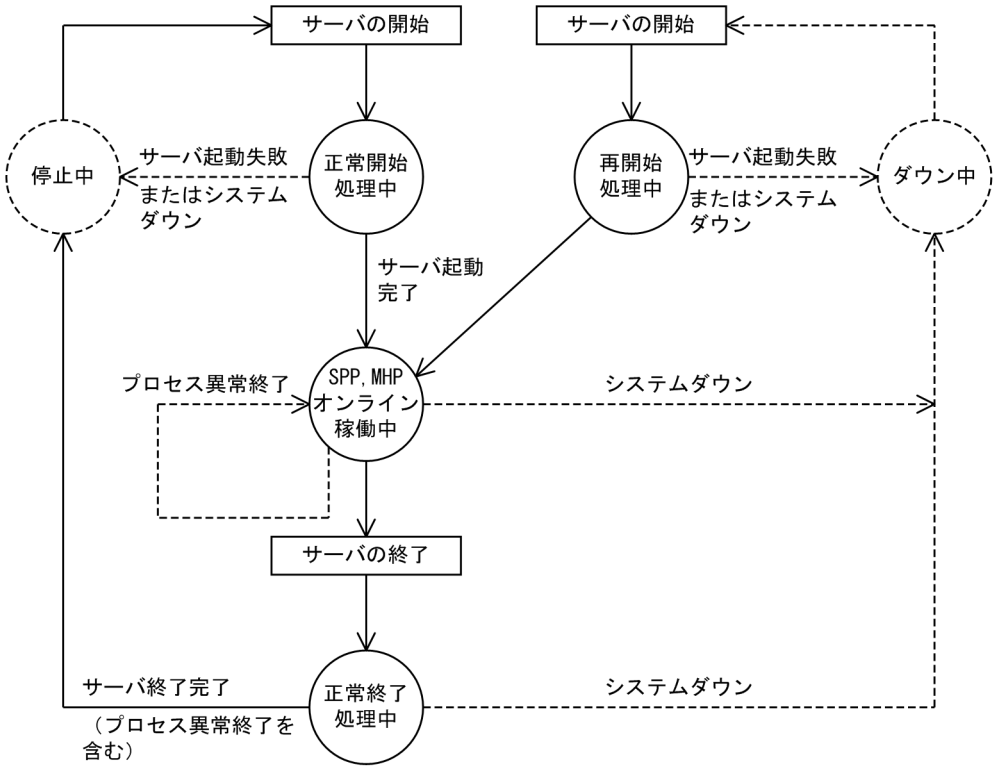


注1　強制正常開始（dcstart -nコマンド）でOpenTP1を開始すると，前回のユーザサーバの状態に関係なく正常開始になります。

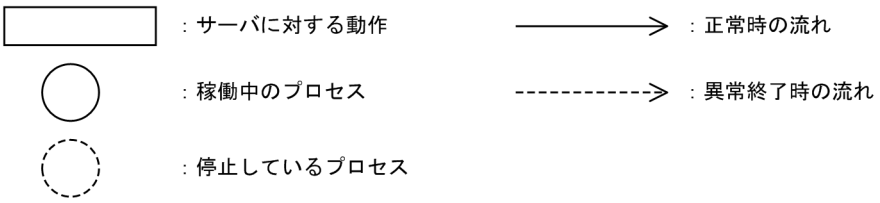
注2　ユーザサーバは，サーバの開始（dcsvstartコマンド），OpenTP1システムの開始（dcstartコマンド），またはOpenTP1システムの自動起動機能で開始します。

注3　ユーザサーバを強制停止すると，OpenTP1システムの異常終了を含むプロセス異常終了と，同じ遷移をします。

図 2-48　ユーザサーバの状態遷移（SPP，MHP）



（凡例）



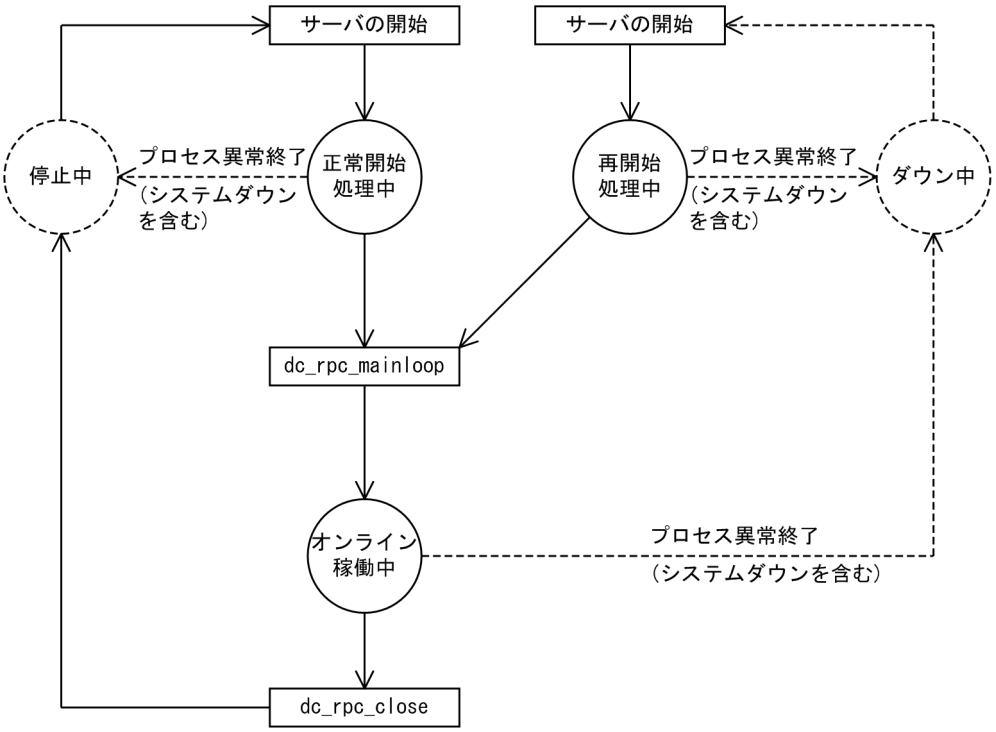
注1　強制正常開始（dcstart -nコマンド）でOpenTP1を開始すると，前回のユーザサーバの状態に関係なく正常開始になります。

注2　ユーザサーバは，サーバの開始（dcsvstartコマンド），OpenTP1システムの開始（dcstartコマンド），またはOpenTP1システムの自動起動機能で開始します。
このとき，常駐プロセス数として指定された数のプロセスを起動し，すべてのプロセスのdc_rpc_mainloop関数，またはdc_mcf_mainloop関数が正常に実行されると，サーバの開始処理が完了します。
dc_rpc_mainloop関数，またはdc_mcf_mainloop関数呼び出しが成功する前に，終了するプロセスが一つでもあると，サーバの開始処理が失敗します。

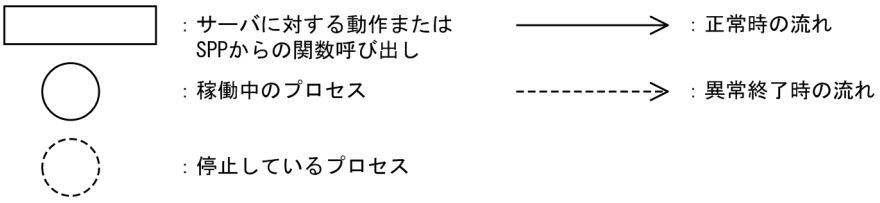
注3　ユーザサーバは，サーバの終了（dcsvstopコマンド），またはOpenTP1システムの終了（dcstopコマンド）で終了します。

注4　ユーザサーバを強制停止すると，OpenTP1システムの異常終了を含むプロセス異常終了と，同じ遷移をします。

図 2-49　ユーザサーバの状態遷移（ソケット受信型サーバ　SPP）



（凡例）



注1　強制正常開始（dcstart -nコマンド）でOpenTP1を開始すると，前回のユーザサーバの状態に関係なく正常開始になります。

注2　ユーザサーバは，サーバの開始（dcsvstartコマンド），OpenTP1システムの開始（dcstartコマンド），またはOpenTP1システムの自動起動機能で開始します。

注3　ユーザサーバを強制停止すると，OpenTP1システムの異常終了を含むプロセス異常終了と，同じ遷移をします。

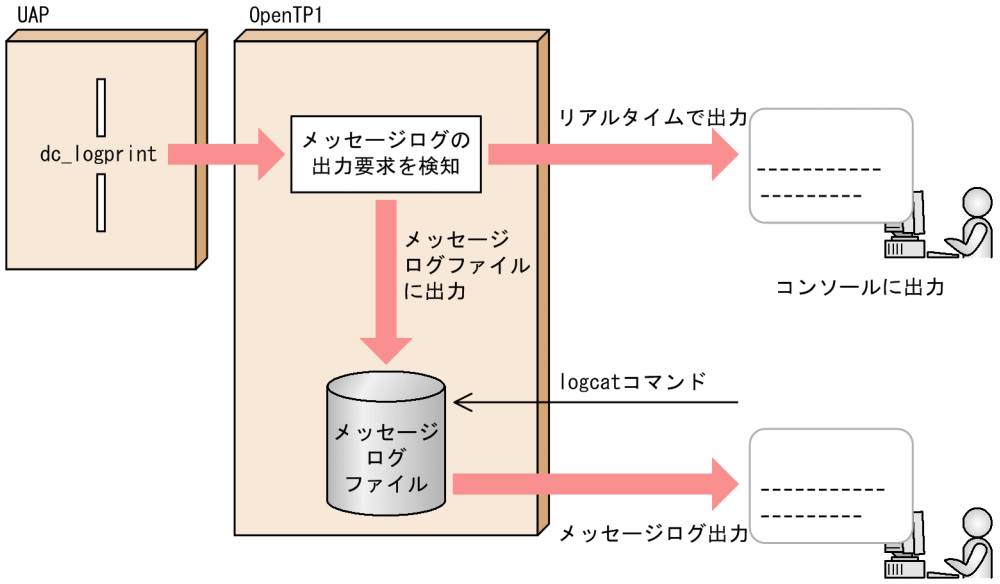
2.5.1　メッセージログをアプリケーションプログラムから出力

UAP から dc_logprint 関数【CBLDCLOG('PRINT ')】を呼び出すと，ユーザ任意の情報を OpenTP1 からのメッセージログとして出力できます。メッセージログは**メッセージログファイル**に出力されます。メッセージログファイルに出力された内容を表示するときは，**logcat コマンド**を実行して，標準出力に出力します。

メッセージログファイルへの出力と同時に，標準出力にリアルタイムに出力させることもできます。標準出力にリアルタイムで出力するかどうかは，**ログサービス定義**で指定します。

UAP からのメッセージログ出力の概要を次の図に示します。

図 2-50　UAP からのメッセージログ出力の概要



(1)　メッセージログとして出力する内容

メッセージログファイルに出力するメッセージログの情報の内容を次の表に示します。UAP から指定する項目は，要求元プログラム ID，メッセージ ID，メッセージログテキストです。メッセージログファイルには次の表で示す情報と OpenTP1 の制御コードが出力されます。

表 2-2　メッセージログファイルに出力するメッセージログの内容

番号	項目		出力される長さ	内容
−	行ヘッダ	メッセージログ通番	半角文字 7 けた	メッセージログ全体の通番です。障害が起こってメッセージログが抜けた場合は，このメッセージログ通番に抜けがあることでわかります。

番号	項目		出力される長さ	内容
−	行ヘッダ	プロセス ID	半角数字 10 けた または，半角数字 5 けた (ログサービス定義の環境変数 DCLOGDEFPID に 1 を指定した場合は，半角数字 5 けた)	メッセージログ出力を指定したプロセスの ID を示します。
		プロセス単位のメッセージログ通番	半角数字 7 けた	出力を要求したプロセス単位の，メッセージログの通番です。
1.	OpenTP1 識別子		半角英数字 2 けた	OpenTP1 システムごとの識別子です。
2.	日時		整数 19 けた	メッセージログの出力を要求した時間です。 「年／月／日　時：分：秒」の形式で出力されます。
3.	要求元ノード名		半角英数字 8 けた	メッセージログの出力を要求した UAP があるノードの名称です。 ノード名の先頭 8 文字が出力されます。
4.	要求元プログラム ID		半角英数字 3 けた	最初の 1 文字は「*」で固定として，あとの 2 文字は UAP で指定した要求元プログラム ID が設定されます。
5.	メッセージ ID		半角英数字 11 けた	メッセージログ出力を要求したときに，UAP でメッセージログごとに付ける識別子です。メッセージ ID の形式は次のとおりです。 KFCA$n_1n_2n_3n_4n_5$-x KFCA： 固定部分です。 $n_1n_2n_3n_4n_5$： UAP で指定する通番です。UAP で出力するメッセージログには，05000 から 06999 までの通番が割り当てられています。 x： メッセージログの種別を英字大文字で付けます。 E…エラーメッセージログ I…通知メッセージログ W…警告メッセージログ R…応答要求メッセージログ
6.	メッセージログテキスト		可変長 最大 222 バイト (Linux 版だけ最大 444 バイト)	UAP が指定した任意の文字列です。

(2) メッセージログの出力形式

UAP から dc_logprint 関数で出力要求したメッセージログを，logcat コマンドで標準出力に表示する形式を次の図に示します。次の図はオプションを省略して入力した例です。logcat コマンドについては，マニュアル「OpenTP1 運用と操作」を参照してください。図中の番号は表 2-2 と対応しています。

図 2-51 メッセージログの出力形式

```
1.          2.           3.    4.     5.                  6.
A  1996/02/19 15：27：41 host1   *sv KFCA05500-I ユーザサービスsvにサービス要求がありました。
A  1996/02/19 15：27：43 host1   *sv KFCA05527-I ユーザサービスsvはトランザクションの処理とし
て起動しています。
```

（凡例）
1. OpenTP1識別子
2. 日時
3. 要求元ノード名
4. 要求元プログラムID
   先頭に＊があることで，UAPから出力要求したメッセージログであることがわかります。
5. メッセージID
   UAPから出力要求したメッセージログには，05000から06999までの値が割り当てられています。
6. メッセージログテキスト
   UAPで設定した文字列です。

(3) NETM にメッセージを渡すときの注意

UAP から出力するメッセージログも，OpenTP1 のメッセージログと同様に，統合ネットワーク管理システム（NETM）の操作支援端末に出力できます。NETM に出力するメッセージログの内容は次のとおりです。

- 行ヘッダの中の項目（ログサービス定義に指定）
- メッセージ ID
- メッセージテキスト

さらに，操作支援端末に出力するメッセージログの表示色を，UAP で設定できます。

UAP から出力するメッセージログを NETM に出力するときは，次のことに気を付けてください。

- NETM に出力するメッセージログは，160 バイト以下になるようにしてください。160 バイトを超えると，NETM でメッセージを分割して VOS3 に引き渡すので，一つのメッセージの行間に別のメッセージが混ざって出力される場合があります。また，メッセージログの長さが 256 バイトを超えると，OpenTP1 のログサービスで 256 バイトを超える部分が切り捨てられます。
- NETM に出力するメッセージテキストの中には，復改文字「¥n」を入れないようにしてください。「¥n」がある場合，NETM でメッセージを「¥n」で分割して VOS3 に引き渡すので，一つのメッセージの行間に別のメッセージが混ざって出力される場合があります。

NETM：Integrated Network Management System

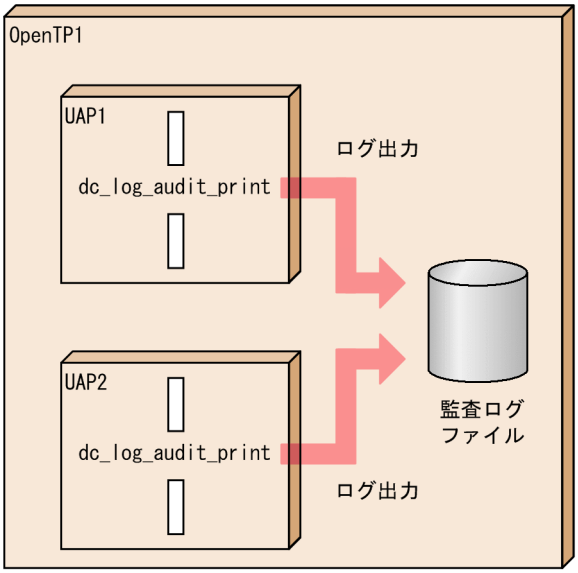
2.6　監査ログの出力

監査ログとは，システム構築者，運用者，および使用者が OpenTP1 のプログラムに対して実行した操作，およびその操作に伴うプログラムの動作の履歴が出力されるファイルです。

OpenTP1 では，UAP に対する操作の実行，UAP 内の処理のタイミングなどで監査ログが出力されます。UAP から任意の監査ログを出力するには，dc_log_audit_print 関数【CBLDCADT('PRINT ')】を呼び出します。

UAP から監査ログが出力される流れを次の図に示します。

図 2-52　UAP から監査ログが出力される流れ



監査ログファイルに出力される監査ログの項目とその内容を次の表に示します。出力される内容を UAP から指定できる項目は，メッセージ ID，発生コンポーネント名，監査事象の種別，監査事象の結果，動作情報，および自由記述です。

表 2-3　監査ログファイルに出力される監査ログの項目

出力の指定	項目	出力される長さ（最大バイト数）	内容
UAP から指定できる項目	メッセージ ID	11	監査ログの識別子
	発生コンポーネント名	3	監査事象が発生したコンポーネントの名称「*AA」の形式で出力されます。AA は dc_log_audit_print 関数で指定した値です。
	監査事象の種別	32	該当する監査事象のカテゴリ名
	監査事象の結果	10	監査事象の結果

出力の指定	項目	出力される長さ（最大バイト数）	内容
UAP から指定できる項目	動作情報	32	該当事象を発生させたサブジェクトが指示したオブジェクトに対する行為（参照，追加，更新，削除など）
	自由記述	1024	監査事象の内容を示す文字列
OpenTP1 によって自動的に指定される項目	ヘッダ情報	12	監査ログのヘッダ情報
	通番	7	監査ログの通番
	日付・時刻	29	監査ログの取得日時
	発生プログラム名	32	監査事象が発生したプログラムの名称
	発生プロセス ID	10	監査事象が発生したプロセスのプロセス ID
	発生場所	255	監査事象が発生したホストの識別情報
	サブジェクト識別情報	256	監査事象を発生させた利用者の識別情報
	オブジェクト情報	256	サービス名 サービス関数内で監査ログが出力される場合だけ，サービス名が出力されます。それ以外の場合は出力されません。
	オブジェクトロケーション情報	64	ユーザサーバ名
	リクエスト送信元ホスト	255	監査事象が複数のプログラム間での連係動作に関連する場合の，リクエスト送信元ホストのホスト識別情報 リクエスト送信元ホストの情報がない場合は出力されません。
	ロケーション識別情報	64	環境変数 DCDIR に設定されたパス名称

2.7　ユーザジャーナルの取得

UAP から任意の情報を，ユーザジャーナル（UJ）としてシステムジャーナルファイルに出力できます。ユーザジャーナルを取得するときは，UAP から dc_jnl_ujput 関数【CBLDCJNL('UJPUT ')】を呼び出します。

ユーザジャーナルを取得する機能は，TP1/Server Base の場合だけ使用できます。TP1/LiNK では，UAP からユーザジャーナルを取得できません。

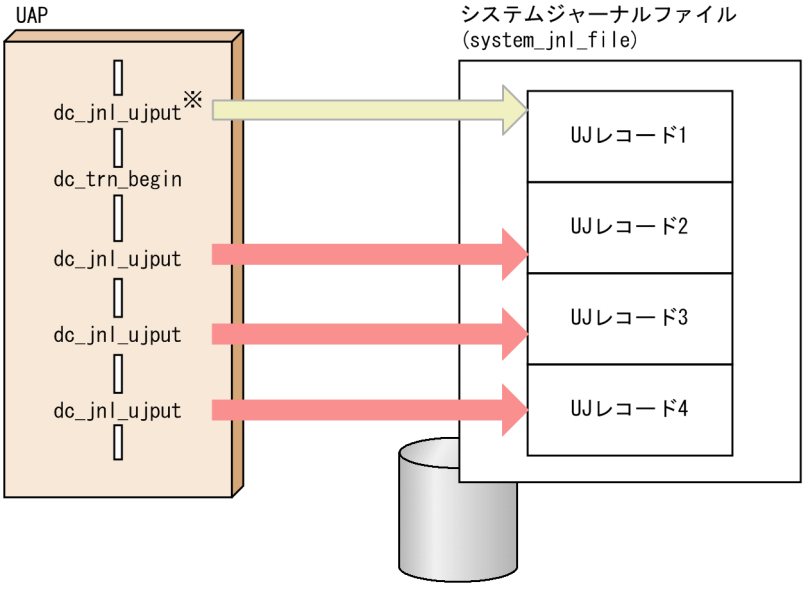
dc_jnl_ujput 関数を呼び出して取得するユーザジャーナルの単位を **UJ レコード**といいます。dc_jnl_ujput 関数を 1 回呼び出すと，UJ レコードを一つ取得できます。

UJ レコードは，トランザクションの範囲外，またはトランザクションの範囲内で取得できます。トランザクションの範囲外で取得する UJ レコードをトランザクション外 UJ と呼び，トランザクションの範囲内で取得する UJ レコードをトランザクション内 UJ と呼びます。トランザクション外 UJ レコードは，ジャーナルバッファに空きがなくなったとき，またはほかのアプリケーションのトランザクションが正常終了した同期点（コミットした時点）で，システムジャーナルファイルに出力されます。

トランザクションが発生しないアプリケーションで UJ レコードを取得する場合は，flags に DCJNL_FLUSH を設定した dc_jnl_ujput 関数を，適切なタイミングで呼び出してください。

ユーザジャーナルの取得を次の図に示します。

図 2-53　ユーザジャーナルの取得



注※
トランザクション外UJレコードは，ジャーナルバッファに空きがなくなったとき，
またはほかのアプリケーションのトランザクションが正常終了した同期点（コミットした時点）で，
システムジャーナルファイルに出力されます。

dc_jnl_ujput 関数を呼び出したトランザクションの処理に障害が起こった場合，ユーザジャーナルを取得した処理はロールバックで無効にはできません。dc_jnl_ujput 関数を呼び出した UAP の処理を部分回復しても，UJ レコードはシステムジャーナルファイルに出力されます。

2.8 ジャーナルデータの編集

jnlrput コマンドの実行結果をリダイレクトして出力したファイル（jnlrput コマンド出力ファイル）を，UAP から関数で編集できます。ジャーナルデータの編集は，COBOL 言語の API でだけできます。C 言語の API はありません。

UAP では，次に示す手順で関数を呼び出します。

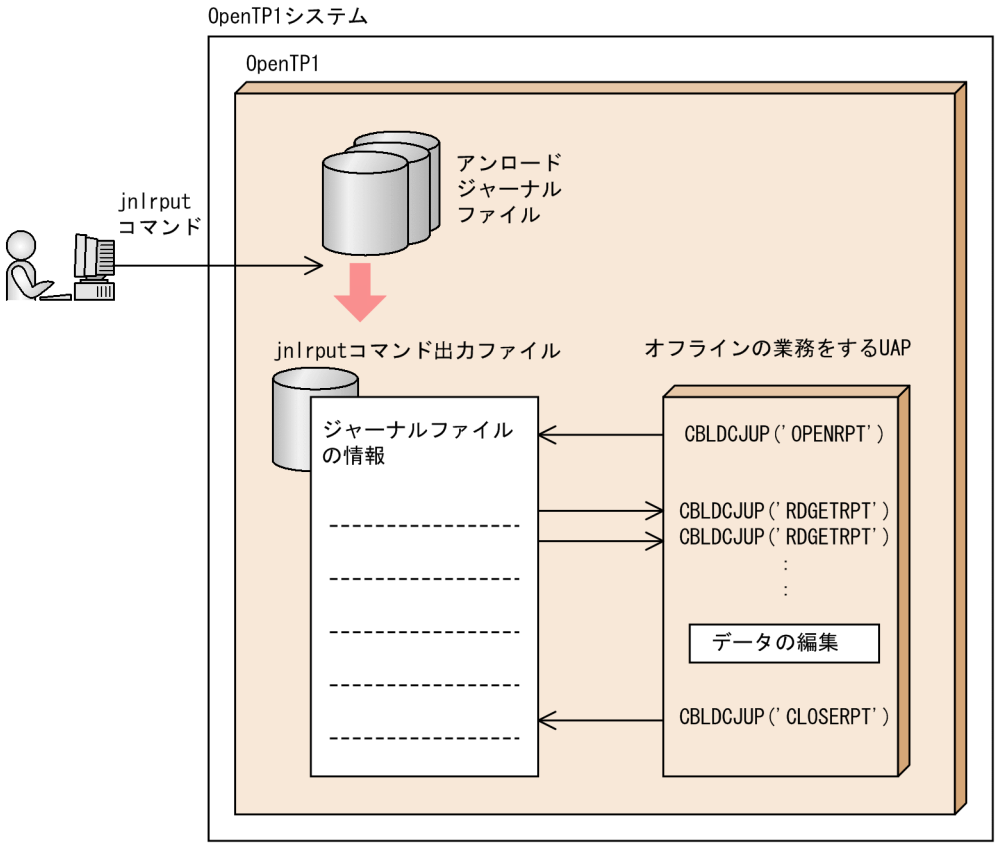
1. jnlrput 出力ファイルを，CBLDCJUP('OPENRPT ')でオープンします。
2. ジャーナルデータを，CBLDCJUP('RDGETRPT')で入力します。ジャーナルデータの種別ごとに一つずつ入力します。必要なジャーナルデータをすべて入力するまで，CBLDCJUP('RDGETRPT')を呼び出します。
3. UAP の処理で編集します。
4. jnlrput 出力ファイルを，CBLDCJUP('CLOSERPT')でクローズします。

ジャーナルデータを編集できるのは，**オフラインの業務をする UAP** だけです。ほかの UAP からは，jnlrput コマンド出力結果ファイルへアクセスしないでください。

ジャーナルデータを編集する機能は，TP1/Server Base の場合だけ使えます。TP1/LiNK では，ジャーナルデータを編集する API は使えません。

ジャーナルデータの編集を次の図に示します。

図 2-54　ジャーナルデータの編集



2.9 メッセージログ通知の受信

OpenTP1 のメッセージログを，システム内に専用に作成するアプリケーションプログラムへ通知できます。通知を受信したアプリケーションプログラムは，他社のネットワーク管理システムへ OpenTP1 の状態を知らせることができます。

メッセージログを通知する場合は，OpenTP1 のログサービス定義の log_notify_out オペランドに Y を指定しておいてください。

2.9.1 メッセージログの通知を受信できるアプリケーションプログラム

メッセージログの通知を受信できるのは，**受信用に作成したアプリケーションプログラム**だけです。OpenTP1 の UAP（SUP，SPP，MHP）では，メッセージログ通知を受信できません。

通知を受信するときは，OpenTP1 の関数を使います。アプリケーションプログラムの作成時には，OpenTP1 のログサービスのヘッダファイルを取り込んで，OpenTP1 のライブラリをリンケージしてください。

通知を受信するアプリケーションプログラムには，OpenTP1 ホームディレクトリを示す環境変数 DCDIR を設定しておいてください。この値は，メッセージログを通知する OpenTP1 と同じ値にしてください。

OpenTP1 のオンライン業務が開始以降のすべてのメッセージログを取得する場合，通知を受信するアプリケーションプログラムは OpenTP1 よりも先に開始しておいてください。

2.9.2 メッセージログの通知の受信手順

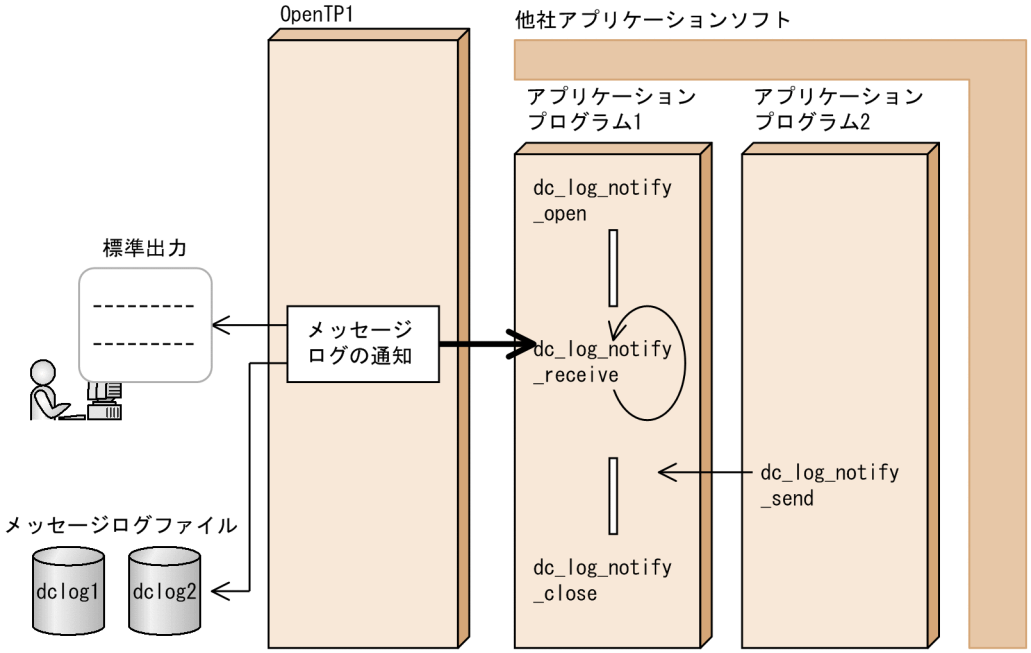
通知を受信するアプリケーションプログラムは，受信の開始を **dc_log_notify_open 関数**で宣言します。そして，**dc_log_notify_receive 関数**でメッセージログを受信します。dc_log_notify_receive 関数で受信できるメッセージログは一つです。複数のメッセージログを受信するには，dc_log_notify_receive 関数を繰り返し呼び出します。

メッセージログの通知の受信を終了するときは，**dc_log_notify_close 関数**を呼び出します。dc_log_notify_close 関数を呼び出したあとで dc_log_notify_open 関数を呼び出せば，再びメッセージログの通知を受信できます。

通知を受信するアプリケーションプログラムは，OpenTP1 が終了したあとでも dc_log_notify_close 関数を呼び出すまで待ち続けます。待ち続けているアプリケーションプログラムに受信処理の終了を知らせる場合には，ほかのアプリケーションプログラムから **dc_log_notify_send 関数**でデータを送信します。受信処理の終了を知らせるアプリケーションプログラムでは，dc_log_notify_send 関数を呼び出す前に dc_log_notify_open 関数を呼び出せません。

メッセージログの通知の受信を次の図に示します。

図 2-55　メッセージログ通知の受信



2.9.3　メッセージログの通知を受信するときの注意

メッセージログの通知を受信するときの注意を，次に示します。

- dc_log_notify_open 関数，dc_log_notify_receive 関数，dc_log_notify_close 関数および dc_log_notify_send 関数は，割り込みルーチンからは実行できません。
- dc_log_notify_receive 関数を呼び出すタイミングによっては，OpenTP1 からのメッセージログの通知を受信できない場合があります。受信できない場合を次に示します。

  1. アプリケーションプログラムが停止している間，またはアプリケーションプログラムが dc_log_notify_open 関数を呼び出す前か dc_log_notify_close 関数を呼び出したあとに OpenTP1 が出力したメッセージログ。
  2. OpenTP1 がメッセージログを通知していても，dc_log_notify_receive 関数を呼び出していない場合に，通知されるメッセージログを退避しておくバッファに空きがなくなった以降のメッセージログ。

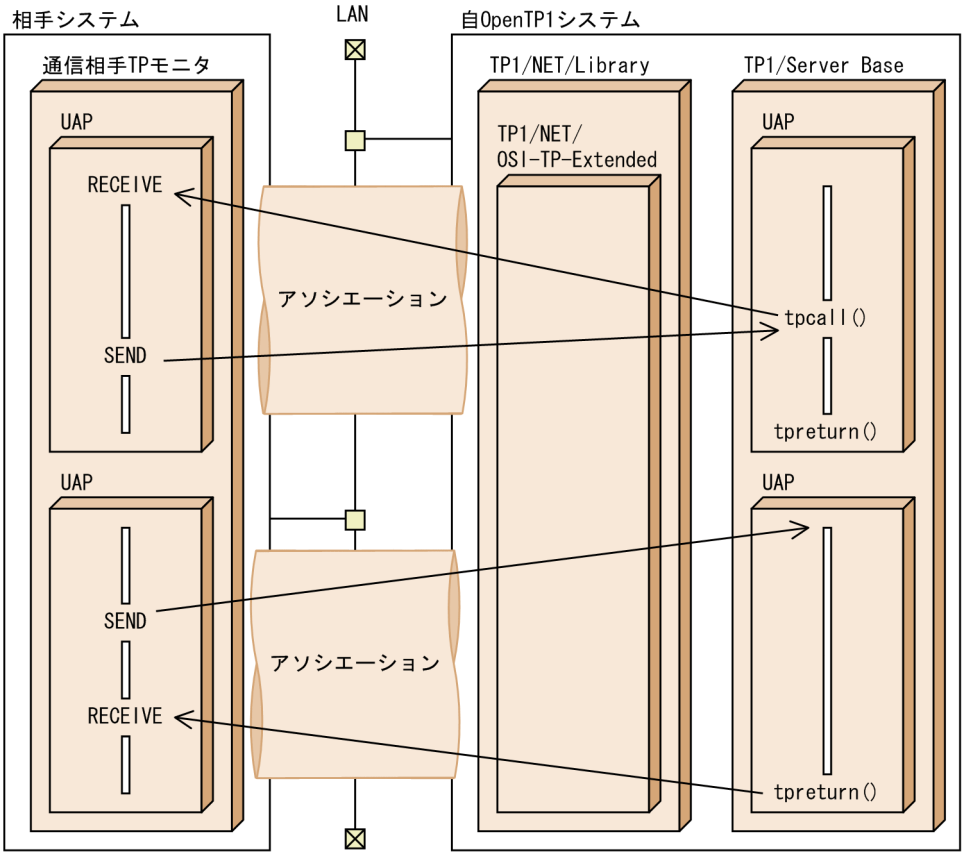
2.10　OSI TP を使ったクライアント／サーバ形態の通信

OpenTP1 のクライアント／サーバ形態の通信では，TCP/IP と OSI TP を通信プロトコルに使えます。ここでは，通信プロトコルに OSI TP を使う場合の概要について説明します。通信プロトコルに OSI TP を使う場合には，TP1/NET/Library，TP1/NET/OSI-TP-Extended および OSI TP の通信管理をする製品が必要です。さらに，OpenTP1 のシステムサービス（XATMI 通信サービス）が必要です。

通信プロトコルに OSI TP を使ったクライアント／サーバ型の通信は，OpenTP1 の基本機能が TP1/Server Base の場合にだけできます。TP1/LiNK では OSI TP 通信はできません。

OSI TP を使ったクライアント／サーバ形態の通信の形態を次の図に示します。

図 2-56　OSI TP を使ったクライアント／サーバ形態の通信の形態



2.10.1　OSI TP 通信で使うアプリケーションプログラム

OpenTP1 の UAP は，相手システムとの通信に XATMI インタフェースを使います。OSI TP を使ったクライアント／サーバ形態の通信で使える OpenTP1 の UAP は，SUP と SPP です。ほかの OpenTP1 の UAP（MHP）は使えません。

UAP では，ノード間の通信プロトコルを意識する必要はありません。

OpenTP1 システムと OpenTP1 システムでは，トランザクション処理を相手システム間で拡張できます。OpenTP1 と OpenTP1 以外のシステムでは，OSI TP を使ってトランザクション処理を相手システム間で拡張できます。

2.10.2　通信イベント処理用 SPP

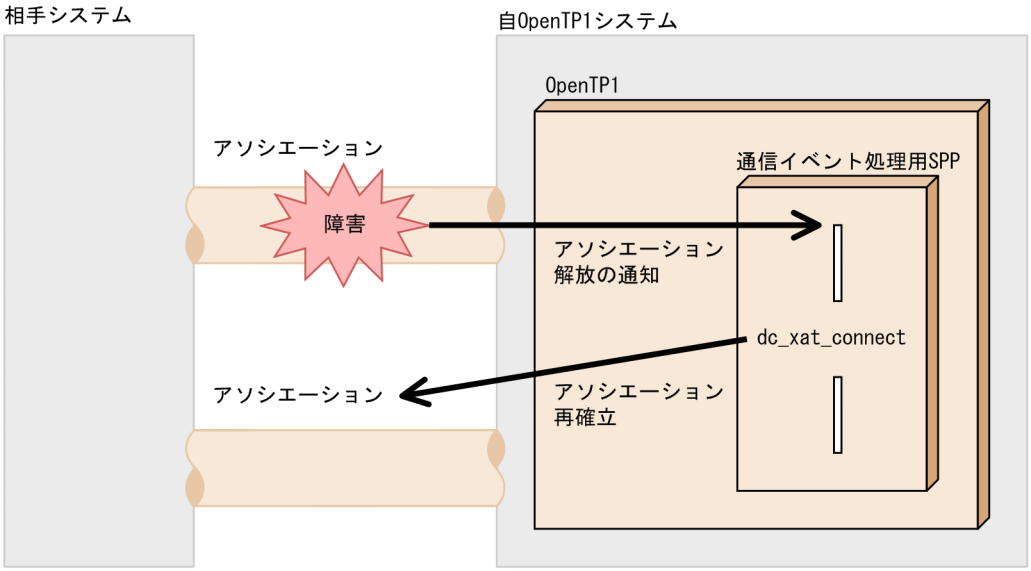
OSI　TP を使ったクライアント／サーバ形態の通信をする場合，アソシエーションの確立と解放を知るための SPP を作成する必要があります。この SPP を**通信イベント処理用 SPP** といいます。通信イベント処理用 SPP を作成すると，アソシエーションの障害解放を通知する通信イベントを受信できます。この通信イベントを受信することで，アソシエーションの再確立のきっかけを知ることができます。また，通信イベント処理用 SPP では，通知された通信イベントの詳細情報から，アソシエーションの属性や状態について知ることができます。

アソシエーションが確立または解放されると，XATMI 通信サービスは非応答型 RPC でサービス要求する様式で，通信イベント処理用 SPP を起動します。通信イベントは，自システムが発呼側か着呼側かどうかに関係なく通知されます。

通信イベント処理用 SPP が受信する情報の内容については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。

通信イベント処理用 SPP の概要を次の図に示します。

図 2-57　通信イベント処理用 SPP の概要



(1)　通信イベント処理用 SPP に関連するシステム定義

通信イベント処理用 SPP が通信イベントを受信できるようにするには，**通信イベント処理用 SPP のサービスグループ名とサービス名**を，あらかじめ **XATMI 通信サービス定義**に指定しておきます。このとき，どのオペランドにサービスグループ名とサービス名を指定するかで，受け取れる通信イベントが異なります。

xat_aso_con_event_svcname オペランド
　アソシエーションの確立通知の通信イベント

xat_aso_discon_event_svcname オペランド
　アソシエーションの正常解放の通信イベント

xat_aso_failure_event_svcname オペランド
　アソシエーションの異常解放の通信イベント

複数のオペランドに同じサービスグループ名とサービス名を指定すると，一つの通信イベント処理用 SPP が複数の通信イベントを受信できるようになります。

通信イベント処理用 SPP のユーザサービス定義の server_type オペランドには，"betran"を指定してください。

(2)　通信イベント処理用 SPP のアソシエーションの確立

通信イベント処理用 SPP から関数を呼び出して，アソシエーションを確立できます。アソシエーションを確立するときは，dc_xat_connect 関数【CBLDCXAT('CONNECT ')】を呼び出します。dc_xat_connect 関数がリターンすると，正常に確立したことを受信する通信イベント処理用 SPP でアソシエーションに関する情報を受け取れます。

dc_xat_connect 関数で確立できるアソシエーションは，自システムが発呼側の場合だけです。また，関数のリターンとアソシエーションの確立は同期しないため，dc_xat_connect 関数を呼び出したサービス関数では，確立を通知する通信イベントは受信できません。

(3)　アソシエーションの状態が通知されるタイミング

通知されるアソシエーションの確立には，次に示す場合があります。

- OpenTP1 システム開始時のアソシエーション確立
- nettactcn コマンドの実行によるアソシエーション確立
- 通信イベント処理用 SPP からの要求によるアソシエーション確立
- 相手システムからのアソシエーション確立

通知されるアソシエーションの解放には，次に示す場合があります。

- nettactcn コマンドの実行によるアソシエーション強制解放
- 下位層の障害によるアソシエーションの解放
- TP1/NET/OSI-TP-Extended の障害によるアソシエーションの解放
- XATMI 通信サービスの障害によるアソシエーションの解放
- アソシエーションの確立の失敗
- 相手システムからのアソシエーションの正常解放

- 相手システムからのアソシエーションの強制解放

2.10.3 OSI TP 通信で障害が起こった場合

OSI TP を使ったクライアント／サーバ形態の通信で障害が起こった場合，サービスを要求した XATMI インタフェースの関数はエラーリターンします。どのリターン値が返るかについては，「OpenTP1 プログラム作成リファレンス」の該当する言語編にある XATMI インタフェースの各関数の注意事項を参照してください。

通信プロトコルで障害が起こった場合の処置については，マニュアル「OpenTP1 プロトコル TP1/NET/OSI-TP-Extended 編」の障害対策の記述を参照してください。

2.11　性能検証用トレースの取得

OpenTP1 で動作する各種サービスの主なイベントでトレース情報を取得しています。これを性能検証用トレース（prf トレース）といいます。性能検証用トレースは，性能検証の効率，およびトラブルシュートの効率を向上させることが目的のトレース情報です。性能検証用トレースには，次のような特徴があります。

- ノードおよびプロセスにわたる場合でもトレースを追うことができます。
- API の単位でなく，内部のイベント単位でトレースを取得できるので，どの処理が性能ネックか検証できます。

なお，この機能は，TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証できませんので，ご了承ください。

UAP からユーザ固有の性能検証用トレースを取得するには，dc_prf_utrace_put 関数【CBLDCPRF('PRFPUT ')】を呼び出します。

また，最新の性能検証用トレースのプロセス内での取得通番を知るには，dc_prf_get_trace_num 関数【CBLDCPRF('PRFGETN ')】を呼び出します。dc_prf_get_trace_num 関数呼び出し元に，最新の性能検証用トレースのプロセス内での取得通番を通知します。

2.12　リアルタイム統計情報の取得

UAP 内の任意区間での処理の実行時間および実行回数を，リアルタイム統計情報として取得できます。オフラインの業務をする UAP では，任意区間でのリアルタイム統計情報は取得できません。

任意区間でのリアルタイム統計情報を取得する場合は，UAP から dc_rts_utrace_put 関数【CBLDCRTS('RTSPUT ')】を呼び出します。

取得する項目は event_id に，取得に関する動作は flags に設定します。flags では，次の表に示す動作を設定できます。

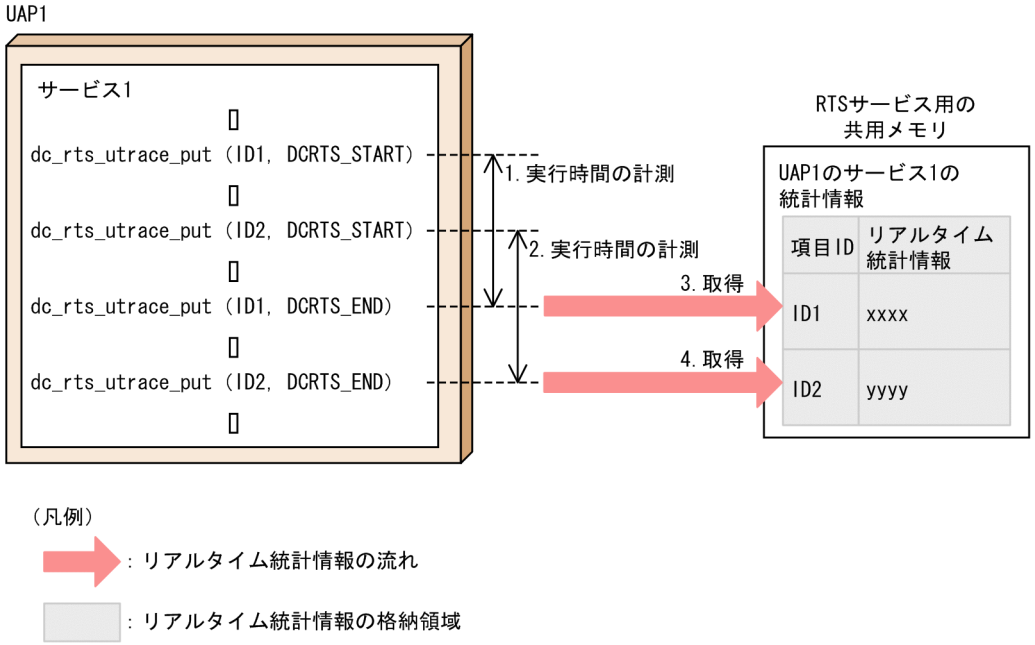
表 2-4　dc_rts_utrace_put 関数の flags の設定

flags の設定	取得に関する動作
DCRTS_START	実行時間の計測を開始
DCRTS_END	実行時間を取得して，計測を終了
DCNOFLAGS	実行回数だけを取得

dc_rts_utrace_put 関数で取得した実行回数および実行時間は，event_id に設定した項目 ID のリアルタイム統計情報として編集，出力します。

任意区間でのリアルタイム統計情報の取得の例を次の図に示します。

図 2-58　任意区間でのリアルタイム統計情報の取得の例



1. 項目 ID1 の実行時間の計測を開始します。
2. 項目 ID2 の実行時間の計測を開始します。

3. 項目 ID1 の実行時間の計測を終了して，実行時間および実行回数の情報を，RTS サービス用の共用メモリに取得します。
4. 項目 ID2 の実行時間の計測を終了して，実行時間および実行回数の情報を，RTS サービス用の共用メモリに取得します。

3 TP1/Message Control を使う場合の機能

メッセージ送受信機能（TP1/Message Control）を使うアプリケーションプログラムで使える機能について説明します。

この章では，各機能を **C 言語の関数名**で説明します。C 言語の関数名に該当する COBOL 言語の API は，関数を最初に説明する個所に【 】で囲んで表記します。それ以降は，C 言語の関数名に統一して説明します。

3.1　MCF 通信サービスに関する運用

ここでは，MCF 通信サービスに関する運用について説明します。運用コマンドによる MCF 通信サービスに関する運用については，マニュアル「OpenTP1 運用と操作」を参照してください。

3.1.1　MCF 通信サービスの状態表示

MCF 通信サービスおよびアプリケーション起動プロセスの状態は，dc_mcf_tlscom 関数【CBLDCMCF('TLSCOM ')】で表示できます。表示内容は MCF 通信サーバ名，MCF 通信サーバのプロセス ID，MCF 通信サービスの状態などです。

3.1.2　API と運用コマンドの機能差異（MCF 通信サービスに関する運用）

MCF 通信サービスに関する運用で使用する関数と運用コマンドの機能差異について，次の表に示します。

表 3-1　関数と運用コマンドの機能差異（MCF 通信サービスに関する運用）

関数名	運用コマンド名	機能差異
dc_mcf_tlscom	mcftlscom	1. すべての MCF 通信サービスの状態を取得します。特定の MCF 通信サービスの状態は取得できません。 2. MCF 通信サービスのプロセス ID は取得できません。 3. MCF 通信サービスの開始待ち合わせはできません。

3.2　コネクションの確立と解放

TP1/Messaging Control を使用して，メインフレームや WS とメッセージ送受信形態で通信する場合，自システムと相手システムとの間に論理的な通信路（**コネクション**）を確立します。

ここでは，UAP からの関数の発行によるコネクションの確立と解放について説明します。運用コマンドによるコネクションの確立と解放については，マニュアル「OpenTP1 運用と操作」を参照してください。

3.2.1　UAP からの関数の発行によるコネクションの確立と解放

コネクションの確立には，dc_mcf_tactcn 関数【CBLDCMCF('TACTCN ')】を使用し，コネクションの解放には，dc_mcf_tdctcn 関数【CBLDCMCF('TDCTCN ')】を使用します。さらに，dc_mcf_tlscn 関数【CBLDCMCF('TLSCN ')】でコネクションの状態を取得できます。

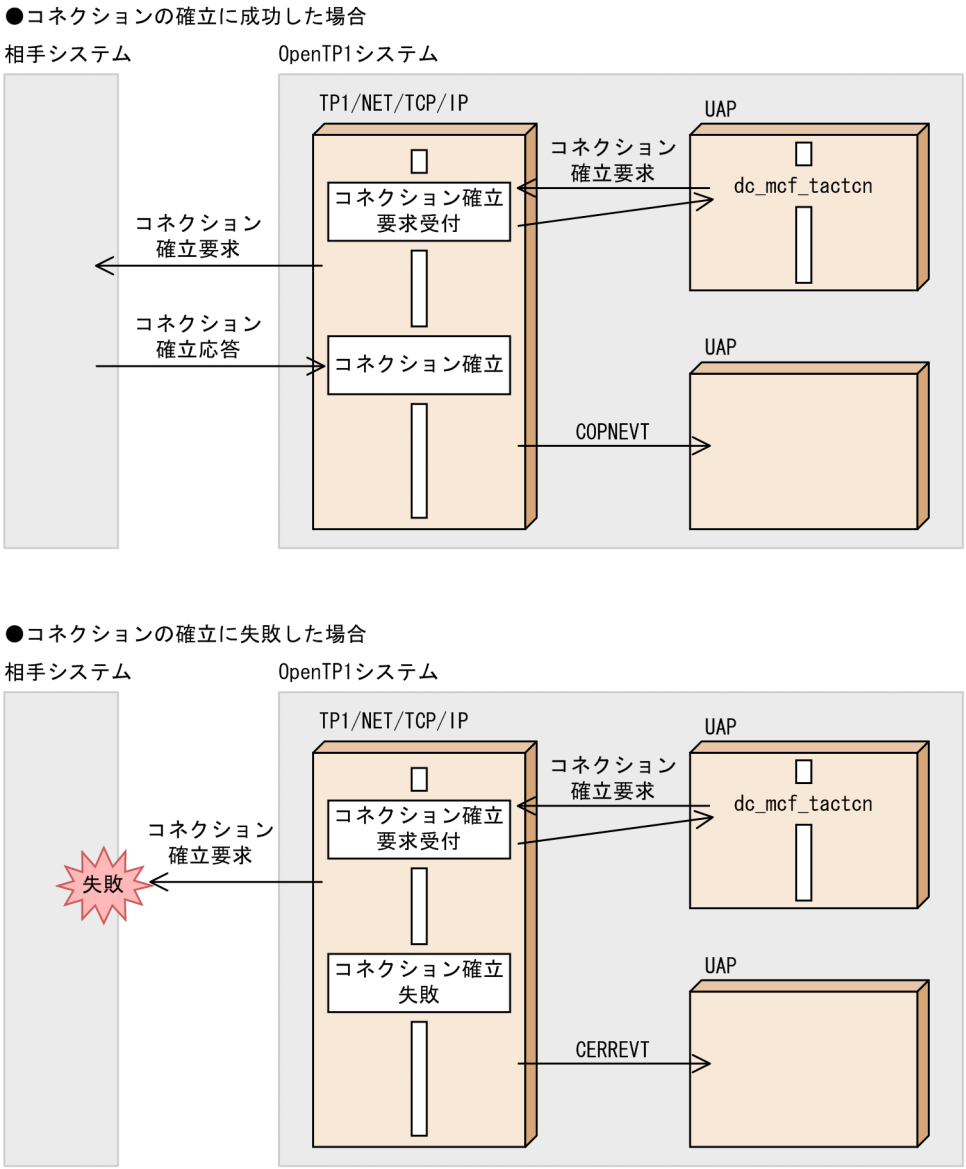
なお，TP1/NET/UDP では，コネクション管理の関数は使用できません。

(1)　コネクションの確立

dc_mcf_tactcn 関数を発行すると，MCF 通信プロセスに相手システムとのコネクション確立を要求します。

使用するプロトコルによっては，相手システムとコネクションが確立されたときやコネクションの確立に失敗したときに，MCF イベントで UAP に通知されます。TP1/NET/TCP/IP 上で dc_mcf_tactcn 関数を使用した場合を例に，コネクションを確立する流れを次の図に示します。

図 3-1　dc_mcf_tactcn 関数を使用したコネクション確立の例

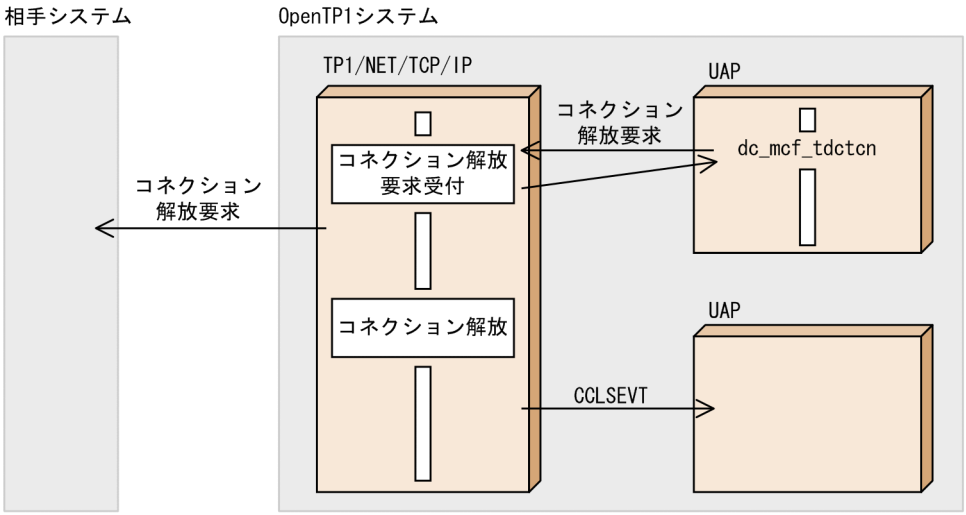


(2)　コネクションの解放

dc_mcf_tdctcn 関数を発行すると，MCF 通信プロセスに相手システムとのコネクション解放を要求します。

使用するプロトコルによっては，コネクションの解放は MCF イベントで UAP に通知されます。TP1/NET/TCP/IP 上で dc_mcf_tdctcn 関数を使用した場合を例に，コネクションを解放する流れを次の図に示します。

図 3-2　dc_mcf_tdctcn 関数を使用したコネクション解放の例



(3)　API と運用コマンドの機能差異（コネクションの確立と解放）

コネクションの確立と解放で使用する関数と運用コマンドの機能差異について，次の表に示します。

表 3-2　関数と運用コマンドの機能差異（コネクションの確立と解放）

関数名	運用コマンド	機能差異
dc_mcf_tactcn	mcftactcn	1. 一つのコネクションに確立を要求します。コネクションの複数指定，および一括指定はできません。 2. コネクショングループへの確立は要求できません。 3. サブコネクションは指定できません。 4. 接続する XP サービスは指定できません。
dc_mcf_tdctcn	mcftdctcn	1. 一つのコネクションに解放を要求します。コネクションの複数指定，および一括指定はできません。 2. コネクショングループへの解放は要求できません。 3. サブコネクションは指定できません。
dc_mcf_tlscn	mcftlscn	1. 一つのコネクションの状態を取得します。コネクションの複数指定，および一括指定はできません。 2. コネクショングループの状態は取得できません。 3. プロトコル種別，コネクション状態だけを取得できます。その他の付加情報やプロトコル固有情報は取得できません。

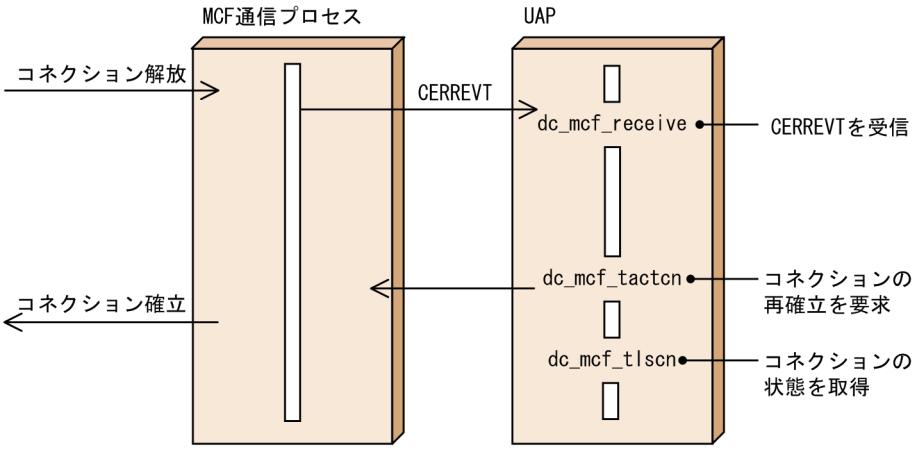
3.2.2　コネクションを再確立・強制解放する場合のコーディング例

ここでは，コネクションを再確立・強制解放する場合の，コーディング例を示します。

(1) コネクションを再確立する場合のコーディング例

CERREVT（コネクション障害）が通知されたあと，コネクションを自動的に再確立する場合の例を，次の図およびコーディング例に示します。

図 3-3　コネクションを自動的に再確立する UAP の例



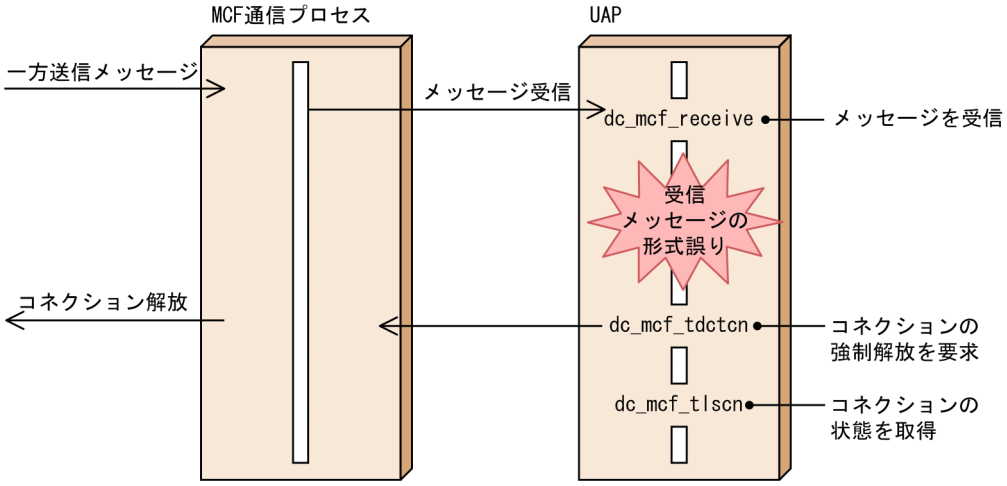

```
void cerrevt(){
    char    rcvdata[256];
    DCLONG  rcv_len;
    DCLONG  rtime;
    int     rtn;
    dcmcf_tactcnopt cnopt;
    dcmcf_tlscnopt  cnopt2;
    DCLONG  infcnt = 1;
    dcmcf_cninf     inf;

    rtn = dc_mcf_receive(DCMCFFRST, DCNOFLAGS, termnam, "", rcvdata,
                         &rcv_len, sizeof(rcvdata), &rtime);
    if (DCMCFRTN_00000 == rtn){

        /* コネクション解放時の処理 */
        /*            :             */

        /* コネクションの再確立要求 */
        memset(&cnopt, 0, sizeof(cnopt));
        strcpy(cnopt.idnam, termnam);

        rtn = dc_mcf_tactcn(DCMCFLE , &cnopt, NULL, NULL, NULL, NULL);
        if (DCMCFRTN_00000 == rtn){
            /* コネクション確立要求の受付：成功 */

            while(1){
                /* コネクションの状態取得 */
                memset(&cnopt2, 0, sizeof(cnopt2));
                strcpy(cnopt2.idnam, termnam);
                memset(&inf, 0, sizeof(inf));

                rtn = dc_mcf_tlscn(DCMCFLE, &cnopt2, NULL, NULL, NULL, &infcnt, &inf, NULL);
                if (DCMCFRTN_00000 == rtn){
                    if (DCMCF_CNST_ACT == inf.status){
```

```
                        /* コネクション確立状態 */
                        break;
                    }
                } else {
                    /* 障害処理 */
                }
                sleep(1);
            }

            /* コネクション確立後の処理 */
            /*          :                */

        } else {
            /* コネクション確立要求の受付：失敗 */
            /* 障害処理 */
        }

    } else {
        /* 障害処理 */
    }

    return;
}
```

(2)　コネクションを強制解放する場合のコーディング例

受信したメッセージに形式誤りがあったときにコネクションを強制解放する場合の例を，次の図およびコーディング例に示します。

図 3-4　コネクションを強制解放する UAP の例




```
void mhprecv(){
    char    rcvdata[256];
    DCLONG  rcv_len;
    DCLONG  rtime;
    int     rtn;
    int     check;
    dcmcf_tdctcnopt cnopt;
    dcmcf_tlscnopt  cnopt2;
    DCLONG  infcnt = 1;
```

```
    dcmcf_cninf     inf;

    rtn = dc_mcf_receive(DCMCFFRST, DCNOFLAGS, termnam, "", rcvdata,
                         &rcv_len, sizeof(rcvdata), &rtime);
    if (DCMCFRTN_00000 == rtn){

        /* 受信メッセージのチェック処理 */
        /*            :                */

        if (0 == check){
            /* チェック結果：正 */
            /* 正常時の処理 */
        } else {
            /* チェック結果：誤 */

            /* コネクションの強制解放要求 */
            memset(&cnopt, 0, sizeof(cnopt));
            strcpy(cnopt.idnam, termnam);

            rtn = dc_mcf_tdctcn(DCMCFLE | DCMCFFRC, &cnopt, NULL, NULL, NULL, NULL);
            if (DCMCFRTN_00000 == rtn){
                /* コネクション強制解放要求の受付：成功 */

                while(1){
                    /* コネクションの状態取得 */
                    memset(&cnopt2, 0, sizeof(cnopt2));
                    strcpy(cnopt2.idnam, termnam);
                    memset(&inf, 0, sizeof(inf));

                    rtn = dc_mcf_tlscn(DCMCFLE, &cnopt2, NULL, NULL, NULL, &infcnt, &inf, NU
LL);
                    if (DCMCFRTN_00000 == rtn){
                        if (DCMCF_CNST_DCT == inf.status){
                            /* コネクション解放状態 */
                            break;
                        }
                    } else {
                        /* 障害処理 */
                    }
                    sleep(1);
                }

                /* コネクション解放後の処理 */
                /*          :               */

            } else {
                /* コネクション強制解放要求の受付：失敗 */
                /* 障害処理 */
            }
        }

    } else {
        /* 障害処理 */
    }

    return;
}
```

3.2.3　コネクションの確立要求の受付開始と終了

コネクションの確立要求の受付を開始するには，dc_mcf_tonln 関数【CBLDCMCF('TONLN ')】を使用します。一方，コネクションの確立要求の受付を終了するには，dc_mcf_tofln 関数【CBLDCMCF('TOFLN ')】を使用します。また，dc_mcf_tlsln 関数【CBLDCMCF('TLSLN ')】で確立要求の受付状態を取得できます。

詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

(1)　API と運用コマンドの機能差異（コネクションの確立要求の受付開始と終了）

コネクションの確立要求の受付開始と終了で使用する関数と運用コマンドの機能差異について，次の表に示します。

表 3-3　関数と運用コマンドの機能差異（コネクションの確立要求の受付開始と終了）

関数名	運用コマンド	機能差異
dc_mcf_tlsln	mcftlsln	1. 対象となる MCF 通信プロセスの MCF 通信プロセス識別子を指定する必要があります。また，すべての MCF 通信プロセスの，サーバ型コネクションの確立要求の受付状態は取得できません。 2. サーバ型コネクションの確立要求の受付状態だけを取得できます。その他の付加情報は取得できません。
dc_mcf_tofln	mcftofln	ありません。
dc_mcf_tonln	mcftonln	ありません。

3.3　アプリケーションに関する運用

ここでは，アプリケーションに関する運用について説明します。運用コマンドによるアプリケーションに関する運用については，マニュアル「OpenTP1 運用と操作」を参照してください。

3.3.1　アプリケーションに関するタイマ起動要求の削除

タイマ起動要求をしたアプリケーションの起動を停止するには，dc_mcf_adltap 関数【CBLDCMCF('ADLTAP ')】を使用します。dc_mcf_adltap 関数を発行すると，指定されたアプリケーションに対するタイマ起動要求を削除し，アプリケーションの起動を停止できます。

3.3.2　API と運用コマンドの機能差異（アプリケーションに関する運用）

アプリケーションに関する運用で使用する関数と運用コマンドの機能差異について，次の表に示します。

表 3-4　関数と運用コマンドの機能差異（アプリケーションに関する運用）

関数名	運用コマンド	機能差異
dc_mcf_adltap	mcfadltap	1. 一つのアプリケーションのタイマ起動要求を削除します。アプリケーションの複数指定，および一括指定はできません。 2. 対象となるアプリケーション起動プロセスの，アプリケーション起動プロセス識別子を指定する必要があります。また，すべてのアプリケーション起動プロセスのタイマ起動要求は削除できません。

3.4　論理端末の閉塞と閉塞解除

ここでは，UAP からの関数の発行による論理端末の閉塞と閉塞解除について説明します。運用コマンドによる論理端末の閉塞と閉塞解除については，マニュアル「OpenTP1 運用と操作」を参照してください。

3.4.1　論理端末の状態表示

論理端末の状態は，dc_mcf_tlsle 関数【CBLDCMCF('TLSLE ')】で表示できます。表示できる内容は，MCF 識別子，論理端末名称，論理端末状態（閉塞状態，または閉塞解除状態）などです。

論理端末の状態は UAP 中で指定した領域に格納されます。

3.4.2　論理端末の閉塞と閉塞解除

論理端末の閉塞状態とは，UAP が送信要求したメッセージを，相手システムに送信できない状態です。この状態で UAP が送信要求した場合，送信要求は正常に受け付けられますが，送信メッセージは出力キューに滞留します。また，この状態のとき，相手システムからの受信メッセージのスケジュールは，正常に行われます。

論理端末を閉塞するには，dc_mcf_tdctle 関数【CBLDCMCF('TDCTLE ')】を使用します。閉塞中の一方送信メッセージの送信要求は，出力キューに滞留します。なお，論理端末は障害によって閉塞することもあります。

一方，論理端末の閉塞解除状態とは，論理端末が持つ機能を使用できる状態です。

論理端末の閉塞を解除するには，dc_mcf_tactle 関数【CBLDCMCF('TACTLE ')】を使用します。閉塞が解除されると，出力キューに残っているメッセージが送信されます。なお，コネクションが未確立の場合は，論理端末の閉塞解除はできません。

3.4.3　論理端末の出力キュー削除

コネクションの確立後，出力キューに残っているメッセージを破棄するには，dc_mcf_tdlqle 関数【CBLDCMCF('TDLQLE ')】を使用します。

dc_mcf_tdlqle 関数を発行すると，ディスクキューとメモリキューの出力キューに残っているすべてのメッセージを削除し，削除したメッセージごとに MCF イベントを起動します。ただし，dc_mcf_tdlqle 関数を発行するためには，あらかじめ mcftdctle コマンドまたは dc_mcf_tdctle 関数で論理端末を閉塞しておく必要があります。

3.4.4　API と運用コマンドの機能差異（論理端末の閉塞と閉塞解除）

論理端末の閉塞と閉塞解除で使用する関数と運用コマンドの機能差異について，次の表に示します。

表 3-5　関数と運用コマンドの機能差異（論理端末の閉塞と閉塞解除）

関数名	運用コマンド	機能差異
dc_mcf_tactle	mcftactle	1. 一つの論理端末に閉塞解除を要求します。論理端末の複数指定，および一括指定はできません。 2. 論理端末の端末状態，およびキュー状態の，両方の閉塞を解除します。どちらか片方だけを指定することはできません。
dc_mcf_tdctle	mcftdctle	1. 一つの論理端末に閉塞を要求します。論理端末の複数指定，および一括指定はできません。 2. 論理端末の端末状態およびキュー状態の両方を閉塞します。どちらか片方だけを指定することはできません。
dc_mcf_tdlqle	mcftdlqle	1. 一つの論理端末に出力キューの削除を要求します。論理端末の複数指定，および一括指定はできません。 2. ディスクキューおよびメモリキューの，両方を削除します。どちらか片方だけを指定することはできません。 3. MCF アプリケーション定義に未処理送信メッセージ廃棄通知イベント（ERREVTA）が定義されている場合，ERREVTA を通知します。通知を抑止することはできません。
dc_mcf_tlsle	mcftlsle	1. 一つの論理端末の状態を取得します。論理端末の複数指定，および一括指定はできません。 2. 論理端末状態だけを取得できます。その他の付加情報は取得できません。 3. 最大未送信メッセージ数をリセットできません。

3.5　通信プロトコル対応製品と運用操作で使える関数

ここでは，通信プロトコル対応製品と運用操作で使える関数の対応について説明します。運用操作とは，次の操作を指します。

- MCF 通信サービスに関する運用
- コネクションの確立と解放
- アプリケーションに関する運用
- 論理端末の閉塞と閉塞解除

通信プロトコル対応製品と運用操作で使える関数の対応を，以降の表に示します。

表 3-6　通信プロトコル対応製品と運用操作で使える関数 1

関数名	通信プロトコル対応製品			
	TP1/NET/OSAS-NIF	TP1/NET/OSI-TP	TP1/NET/SLU - TypeP2	TP1/NET/TCP/IP
dc_mcf_adltap	○	○	○	○
dc_mcf_tactcn	○	○	○	○
dc_mcf_tactle	○	×	○	○
dc_mcf_tdctcn	○	○	○	○
dc_mcf_tdctle	○	×	○	○
dc_mcf_tdlqle	○	×	○	○
dc_mcf_tlscn	○	○	○	○
dc_mcf_tlscom	○	○	○	○
dc_mcf_tlsle	○	×	○	○
dc_mcf_tlsln	×	×	×	○
dc_mcf_tofln	×	×	×	○
dc_mcf_tonln	×	×	×	○

（凡例）

○：使えます。

×：使えません。

表 3-7　通信プロトコル対応製品と運用操作で使える関数 2

関数名	通信プロトコル対応製品		
	TP1/NET/User Agent	TP1/NET/UDP	TP1/NET/XMAP3
dc_mcf_adltap	○	○	○

関数名	通信プロトコル対応製品		
	TP1/NET/User Agent	TP1/NET/UDP	TP1/NET/XMAP3
dc_mcf_tactcn	○	×	○
dc_mcf_tactle	○	○	○
dc_mcf_tdctcn	○	×	○
dc_mcf_tdctle	○	○	○
dc_mcf_tdlqle	○	○	○
dc_mcf_tlscn	○	×	○
dc_mcf_tlscom	○	○	○
dc_mcf_tlsle	○	○	○
dc_mcf_tlsln	×	×	×
dc_mcf_tofln	×	×	×
dc_mcf_tonln	×	×	×

（凡例）

○：使えます。

×：使えません。

3.6　メッセージ送受信

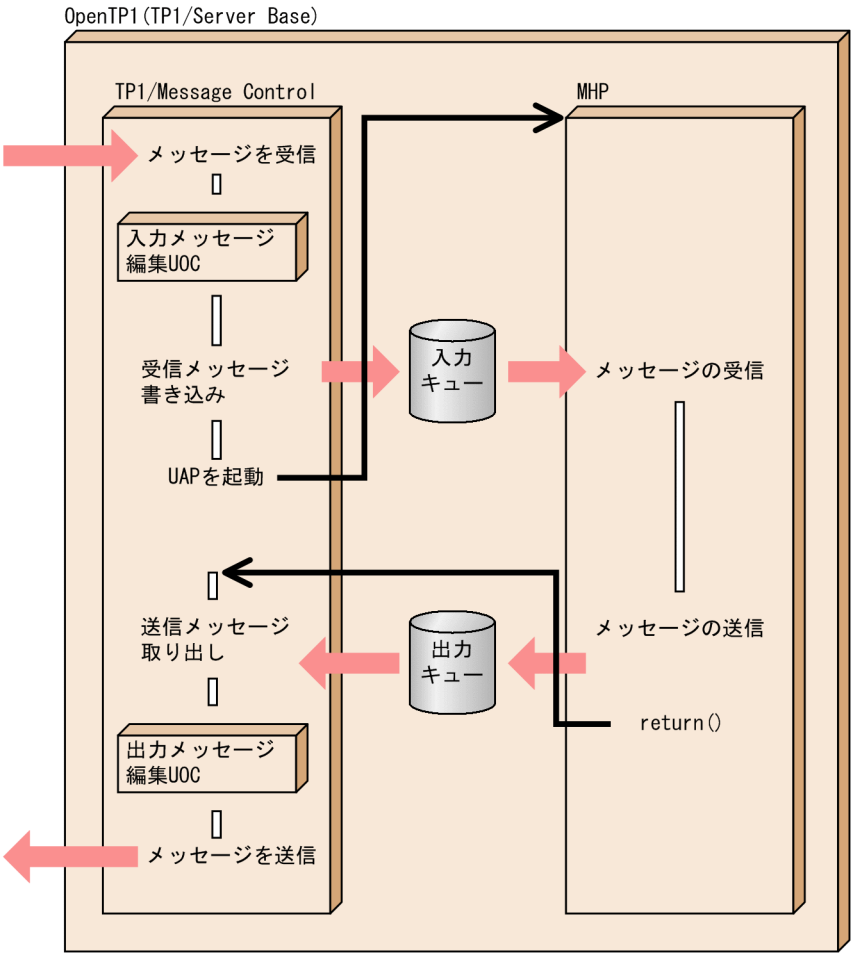
OpenTP1 の基本機能（TP1/Server Base）に TP1/Message Control を組み込んで使うと，広域ネットワーク（WAN）や TCP/IP，および従来型のネットワークを介して，メインフレームや WS とメッセージ送受信形態で通信できます。

メッセージを使って通信する場合は，MHP を使います。また，一部のメッセージ処理は SPP でも使えます。

メッセージ送受信機能は，システムに TP1/Message Control が組み込まれていることが前提となります。また，OpenTP1 の基本機能が TP1/Server Base の場合だけ使えます。TP1/LiNK で MHP を作成するときは，TP1/Messaging が必要です。

メッセージ送受信形態の通信の概要を次の図に示します。

図 3-5　メッセージ送受信形態の通信の概要



UOC について

UAP によるメッセージの処理を，より多様な業務に対応させるために，ユーザオウンコーディング（UOC）を作成できます。UOC は業務に合わせて任意に作成します。

UOC の概要については，「3.9　ユーザオウンコーディング（UOC）」を参照してください。

3.6.1　メッセージの通信形態

(1)　MHP で使えるメッセージの通信形態

MHP で使えるメッセージの通信形態を次に示します。使えるメッセージの通信形態は，通信プロトコル別で異なります。

- 問い合わせ応答形態

  相手システムから dc_mcf_receive 関数【CBLDCMCF('RECEIVE ')】でメッセージを受け取って，dc_mcf_reply 関数【CBLDCMCF('REPLY ')】で応答メッセージを返す形態です。

- 非問い合わせ応答形態（一方受信形態）

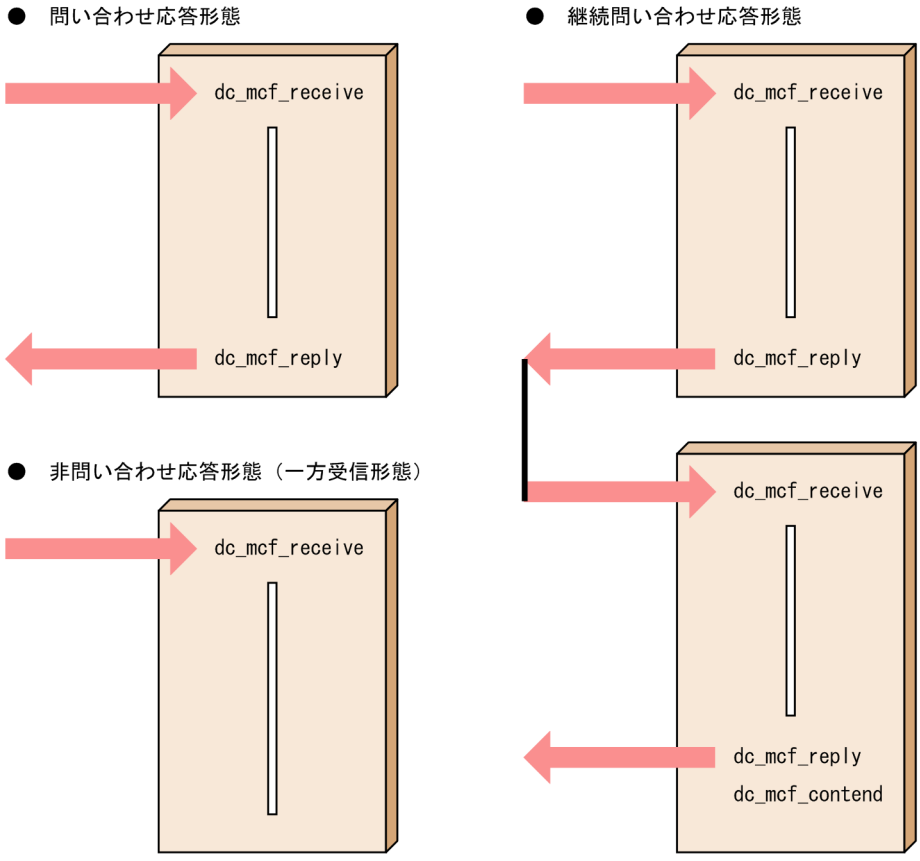
  相手システムから dc_mcf_receive 関数でメッセージを受け取ったあとに，応答メッセージを返さない形態です。

- 継続問い合わせ応答形態

  問い合わせ応答形態を連続させる形態です。dc_mcf_receive 関数でメッセージを受け取って，dc_mcf_reply 関数で応答メッセージを返してから，続けて問い合わせ応答の処理をします。継続問い合わせ応答は dc_mcf_contend 関数【CBLDCMCF('CONTEND ')】で終了させます。

メッセージの通信形態を次の図に示します。

図 3-6　メッセージの通信形態



(2) MHP の各通信形態，および SPP で任意に使えるメッセージ通信機能

MHP と SPP で任意に使えるメッセージ通信機能を次に示します。

- **分岐送信**
  自システムから dc_mcf_send 関数【CBLDCMCF('SEND ')】でメッセージを送信できます。
- **同期送信，同期受信，同期送受信**
  相手システムからのメッセージ受信後に，メッセージを同期的に送信（dc_mcf_sendsync 関数【CBLDCMCF('SENDSYNC')】），同期的に受信（dc_mcf_recvsync 関数【CBLDCMCF('RECVSYNC')】），および同期的に送受信（dc_mcf_sendrecv 関数【CBLDCMCF('SENDRECV')】）できます。送信処理または受信処理が完了するまで，関数はリターンしません。

SPP から dc_mcf_send 関数を呼び出す場合は，その SPP の処理はトランザクションとして稼働していることが前提です。

(3) メッセージの通信形態とアプリケーションの型

メッセージ送受信機能を使う MHP は，使うメッセージの通信形態によって**アプリケーションの型**を指定しておきます。アプリケーションの型は，MCF アプリケーション定義アプリケーション属性定義（mcfaalcap）の type オペランドで指定します。アプリケーションの型には次の三つがあります。

- **応答型（ans 型）**：問い合わせ応答形態の MHP。
- **非応答型（noans 型）**：非問い合わせ応答形態の MHP。
- **継続問い合わせ応答型（cont 型）**：継続問い合わせ応答形態の MHP。

dc_mcf_receive 関数で受信したメッセージを，dc_mcf_send 関数で入力元の論理端末に送信する形態も，noans 型とします。

MHP には，メッセージを処理する形態に合ったアプリケーションの型を指定してください。

指定したアプリケーションの型とメッセージを処理する形態に矛盾がある場合は，メッセージ送受信の関数がエラーリターンするか，または MHP の処理がロールバックされます。矛盾がある例を次に示します。

- 応答型の MHP で，dc_mcf_reply 関数を使わないで終了した場合，またはほかの応答型の MHP を dc_mcf_execap 関数【CBLDCMCF('EXECAP ')】で起動しないで終了した場合
- 非応答型の MHP で dc_mcf_reply 関数を使った場合（非応答型の MHP からの問い合わせ応答をしない（UAP 共通定義（mcfmuap）の-c オプションの noansreply オペランドに no を指定）場合）

MCF イベント処理用 MHP のアプリケーションの型は，通知された MCF イベントによって決まります。MCF イベント処理用 MHP のアプリケーションの型については，「3.10　MCF イベント」を参照してください。

アプリケーションの型と使えるメッセージ送受信関数を次の表に示します。

表 3-8　アプリケーションの型と使えるメッセージ送受信関数

メッセージの形態	アプリケーションの型	メッセージを処理する関数						
		receive	send	reply	sendrecv	recvsync	sendsync	tempput, tempget, contend
問い合わせ応答形態	ans 型	◎	○	◎	△※1	△※1	−	−
非問い合わせ応答形態（一方受信形態）	noans 型	◎	○	○※2	△	△	△	−
継続問い合わせ応答形態	cont 型	◎	○	◎	−	−	−	○

（凡例）

◎：必ず使います。

○：使えます。

△：プロトコル製品によって異なります。詳細については「3.6.1(4)　通信プロトコル対応製品と通信形態で使える関数」を参照してください。

−：使えません。

注※1

TP1/NET/User Agent または TP1/NET/OSAS-NIF を使った場合に呼び出せます。ただし，dc_mcf_recvsync 関数は，複数セグメントを送信した dc_mcf_sendrecv 関数を呼び出したときに，先頭セグメント以外のセグメントを受け取る目的にだけ使えます。

注※2

非応答型の MHP からの問い合わせ応答をする（UAP 共通定義（mcfmuap）の-c オプションの noansreply オペランドに yes を指定）場合に呼び出せます。

(4)　通信プロトコル対応製品と通信形態で使える関数

通信プロトコル対応製品と通信形態別で使える関数の対応を，以降の表に示します。

表 3-9　通信プロトコル対応製品と通信形態で使える関数 1

関数名	通信プロトコル対応製品とアプリケーションの型								
	TP1/NET/OSAS-NIF			TP1/NET/OSI-TP			TP1/NET/SLU - TypeP2		
	noans 型	ans 型	cont 型	noans 型	ans 型	cont 型	noans 型	ans 型	cont 型
dc_mcf_commit	○	×	−	○	−	−	○	−	−
dc_mcf_receive※	○	○	−	○	−	−	○	−	−
dc_mcf_execap	○	○	−	×	−	−	○	−	−
dc_mcf_reply※	△	○	−	×	−	−	×	−	−

関数名	通信プロトコル対応製品とアプリケーションの型								
	TP1/NET/OSAS-NIF			TP1/NET/OSI-TP			TP1/NET/SLU - TypeP2		
	noans 型	ans 型	cont 型	noans 型	ans 型	cont 型	noans 型	ans 型	cont 型
dc_mcf_rollback	○	○	−	○	−	−	○	−	−
dc_mcf_send※	○	○	−	×	−	−	○	−	−
dc_mcf_resend※	○	○	−	×	−	−	○	−	−
dc_mcf_sendrecv※	○	○	−	○	−	−	○	−	−
dc_mcf_sendsync※	×	×	−	○	−	−	×	−	−
dc_mcf_recvsync※	□	□	−	○	−	−	□	−	−
dc_mcf_contend	×	×	−	×	−	−	×	−	−
dc_mcf_tempget	×	×	−	×	−	−	×	−	−
dc_mcf_tempput	×	×	−	×	−	−	×	−	−

（凡例）

○：該当する通信プロトコル対応製品で使えます。

×：使えません。

□：通信プロトコル対応製品で固有な使い方をします。

△：非応答型の MHP からの問い合わせ応答をする（UAP 共通定義（mcfmuap）の-c オプションの noansreply オペランドに yes を指定）場合に使えます。

−：該当する通信プロトコル対応製品では使えない通信形態です。

注※

通信プロトコル対応製品によって，関数の使い方が異なる場合があります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

表 3-10　通信プロトコル対応製品と通信形態で使える関数 2

関数名	通信プロトコル対応製品とアプリケーションの型					
	TP1/NET/TCP/IP			TP1/NET/User Agent		
	noans 型	ans 型	cont 型	noans 型	ans 型	cont 型
dc_mcf_commit	○	×	×	○	×	−
dc_mcf_receive※	○	○	○	○	○	−
dc_mcf_execap	○	○	○	○	○	−
dc_mcf_reply※	△	○	○	△	○	−
dc_mcf_rollback	○	○	○	○	○	−
dc_mcf_send※	○	○	○	○	○	−
dc_mcf_resend※	○	○	○	○	○	−

関数名	通信プロトコル対応製品とアプリケーションの型					
	TP1/NET/TCP/IP			TP1/NET/User Agent		
	noans 型	ans 型	cont 型	noans 型	ans 型	cont 型
dc_mcf_sendrecv※	○	×	×	○	○	−
dc_mcf_sendsync※	○	×	×	×	×	−
dc_mcf_recvsync※	○	×	×	□	□	−
dc_mcf_contend	×	×	○	×	×	−
dc_mcf_tempget	×	×	○	×	×	−
dc_mcf_tempput	×	×	○	×	×	−

（凡例）

○：該当する通信プロトコル対応製品で使えます。

×：使えません。

□：通信プロトコル対応製品で固有な使い方をします。

△：非応答型の MHP からの問い合わせ応答をする（UAP 共通定義（mcfmuap）の-c オプションの noansreply オペランドに yes を指定）場合に使えます。

−：該当する通信プロトコル対応製品では使えない通信形態です。

注※

通信プロトコル対応製品によって，関数の使い方が異なる場合があります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

表 3-11　通信プロトコル対応製品と通信形態で使える関数 3

関数名	通信プロトコル対応製品とアプリケーションの型					
	TP1/NET/UDP			TP1/NET/XMAP3		
	noans 型	ans 型	cont 型	noans 型	ans 型	cont 型
dc_mcf_commit	○	−	−	○	×	×
dc_mcf_receive※	○	−	−	○	○	○
dc_mcf_execap	○	−	−	○	○	○
dc_mcf_reply※	×	−	−	△	○	○
dc_mcf_rollback	○	−	−	○	○	○
dc_mcf_send※	○	−	−	○	○	○
dc_mcf_resend※	○	−	−	○	○	○
dc_mcf_sendrecv※	×	−	−	×	×	×
dc_mcf_sendsync※	○	−	−	×	×	×
dc_mcf_recvsync※	×	−	−	×	×	×

関数名	通信プロトコル対応製品とアプリケーションの型					
	TP1/NET/UDP			TP1/NET/XMAP3		
	noans 型	ans 型	cont 型	noans 型	ans 型	cont 型
dc_mcf_contend	×	−	−	×	×	○
dc_mcf_tempget	×	−	−	×	×	○
dc_mcf_tempput	×	−	−	×	×	○

（凡例）

○：該当する通信プロトコル対応製品で使えます。

×：使えません。

△：非応答型の MHP からの問い合わせ応答をする（UAP 共通定義（mcfmuap）の-c オプションの noansreply オペランドに yes を指定）場合に使えます。

−：該当する通信プロトコル対応製品では使えない通信形態です。

注※

通信プロトコル対応製品によって，関数の使い方が異なる場合があります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

3.6.2　メッセージの構造

メッセージの構造について説明します。

(1)　論理メッセージとセグメント

システム間で通信する上で意味を持つデータの単位を**論理メッセージ**といいます。論理メッセージは，一つまたは複数の**セグメント**から構成されます。セグメントとは，UAP の処理からライブラリ関数の呼び出し 1 回で処理できる単位です。

論理メッセージが一つのセグメントから構成されるときは，関数呼び出し 1 回でメッセージを処理できます。論理メッセージが複数のセグメントから構成されるときは，セグメントの数だけ関数を呼び出して，メッセージを処理します。

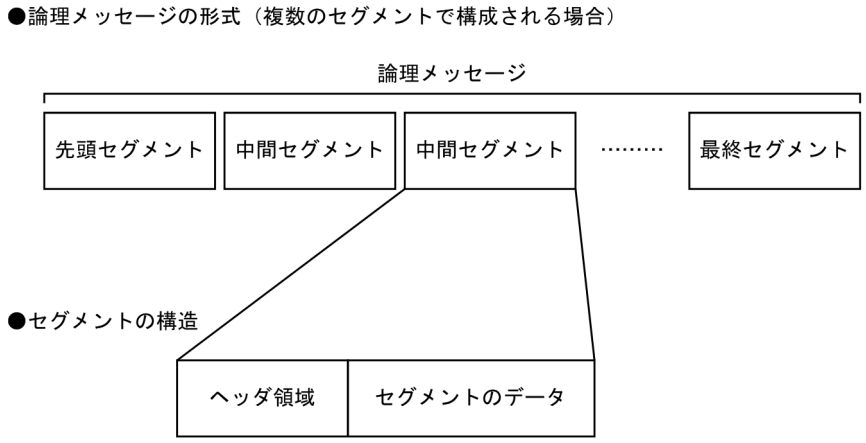
(2)　セグメントの構造

セグメントは，**MCF で使うヘッダ領域**と**セグメントのデータ**から構成されます。ヘッダ領域には，長さによって**バッファ形式 1** と**バッファ形式 2** の 2 種類があります。どちらの形式のヘッダ領域を使うかは，ユーザ任意です。ただし，TP1/NET/XMAP3 を使う場合は，バッファ形式 2 だけを使えます。

ヘッダ領域の長さは，通信プロトコル対応製品によって異なります。ヘッダ領域の長さについては，マニュアル「OpenTP1 プロトコル」のメッセージ送受信 API の説明を参照してください。

論理メッセージとセグメントの関係を次の図に示します。

図 3-7　論理メッセージとセグメントの関係



3.6.3　メッセージの受信

他システムから送信されたメッセージを最終セグメントまで受信すると，MCF は**アプリケーション名**に該当する MHP にメッセージを渡します。MHP ではメッセージを dc_mcf_receive 関数【CBLDCMCF('RECEIVE ')】で受信して処理を開始します。このメッセージ受信を**非同期型のメッセージ受信**といいます。

dc_mcf_receive 関数では，メッセージをセグメント単位で受信します。

メッセージが一つのセグメント（単一セグメント）から構成される場合は，dc_mcf_receive 関数を 1 回だけ呼び出します。

メッセージが複数のセグメントから構成されるときは，dc_mcf_receive 関数をセグメントの数だけ呼び出して受信します。MHP では，メッセージを先頭セグメントから受信して，次に中間以降のセグメントを受信します。中間以降のセグメントを受信し続けると，受信するセグメントがないことを知らせるリターン値が返るので，最終セグメントまで受信し終わったことがわかります。

UOC を使うと，MHP にメッセージを渡す前にメッセージを編集したり，アプリケーション名を変更したりできます。

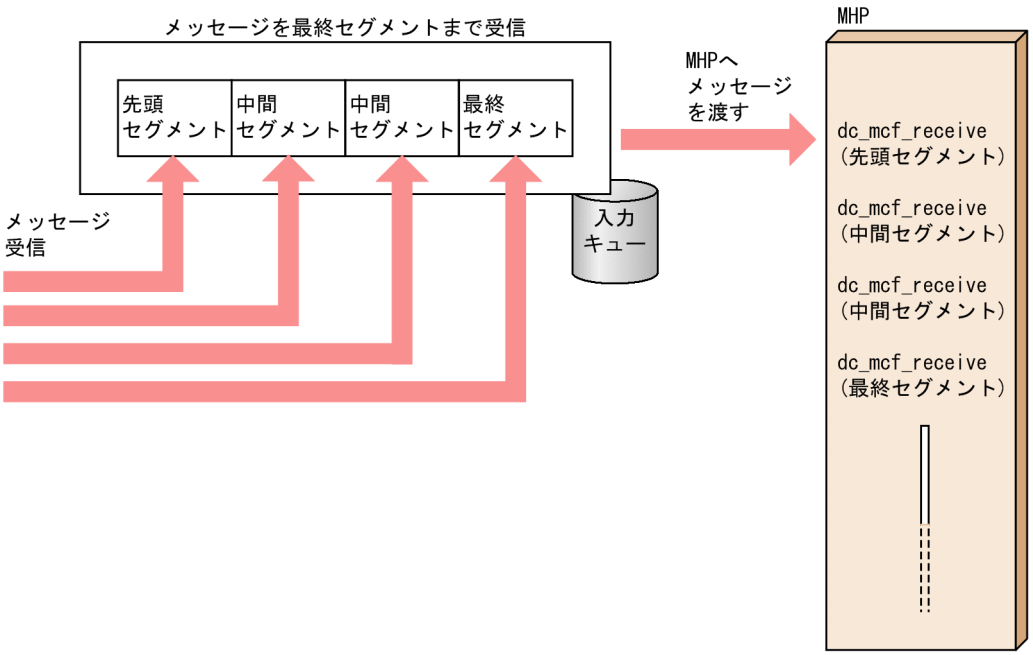
アプリケーション名

アプリケーション名は，メッセージの先頭から，空白の手前までの 1 から 8 バイトの英数字です。先頭から 9 バイト目までに空白がないとき，または先頭が空白のときは，間違ったアプリケーション名と見なします。

アプリケーション名は，入力メッセージ編集 UOC で編集できます。

メッセージの受信を次の図に示します。

図 3-8　メッセージの受信



3.6.4　メッセージの送信

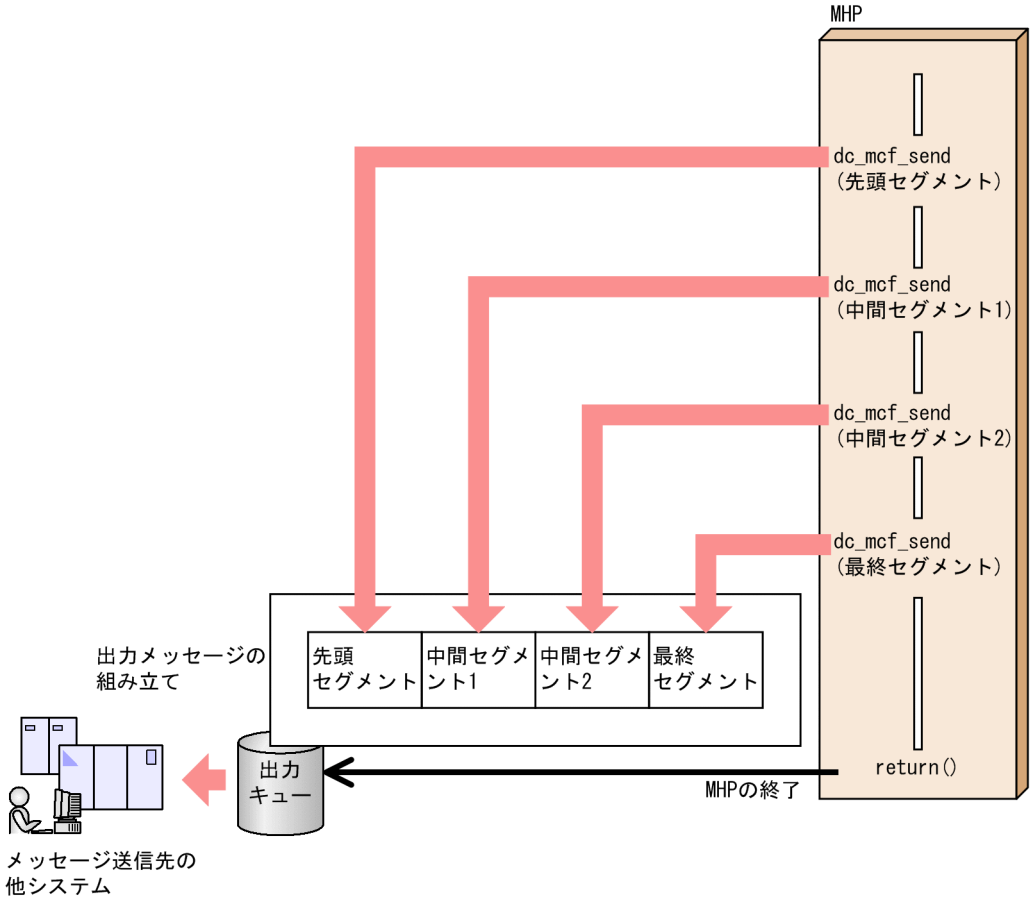
OpenTP1 では，セグメントを送信する UAP のすべての処理が終了したあとで（MHP の終了，または SPP のトランザクション正常終了），それまで UAP から送信したセグメントを一括してメッセージとして送信します。これを**非同期型のメッセージ送信**といいます。一方送信メッセージは dc_mcf_send 関数【CBLDCMCF('SEND ')】，応答メッセージは dc_mcf_reply 関数【CBLDCMCF('REPLY ')】を使います。

非同期型のメッセージ送信では，関数がセグメントを送信したあとに UAP のプロセスが異常終了，またはメッセージ処理ができなくてロールバックとなった場合，それまでに UAP から出力していたセグメントの送信はすべて無効となります。

UAP からセグメントを送信してから実際に出力される前に，UOC で出力通番の編集や出力メッセージの編集など，任意の処理ができます。

メッセージの送信を次の図に示します。

図 3-9　メッセージの送信（非同期型のメッセージ送信）



3.6.5　同期型のメッセージ処理

MHP の処理の途中でメッセージの送信完了を確認したいときや，UAP のメッセージ送受信の処理をシステム間で同期を取りたいときに，同期型のメッセージを使います。同期型のメッセージ送受信では，送信または受信を要求して，その処理がすべて完了してから，UAP から呼び出した関数がリターンします。

(1)　同期型のメッセージの種類

同期型のメッセージ送受信には，送信，および受信をそれぞれ単独でする関数と，送受信を連続してする関数があります。

- **同期型のメッセージ送信**

  同期型のメッセージ送信をするときは dc_mcf_sendsync 関数【CBLDCMCF('SENDSYNC')】を使います。UAP が dc_mcf_sendsync 関数を呼び出すと，MCF はメッセージを出力用のバッファ（メモリ上の出力キュー）に書き込んで，相手システムへ送信します。相手システムへのメッセージ送信が完了したことを MCF で確認したあと，dc_mcf_sendsync 関数はリターンします。

- **同期型のメッセージ受信**

相手システムからのメッセージを受信すると，MCF はメッセージを入力用のバッファに格納します。MHP では，このメッセージを受け取るために dc_mcf_recvsync 関数【CBLDCMCF('RECVSYNC')】を呼び出します。

相手システムからのメッセージを受信していた場合は，dc_mcf_recvsync 関数にメッセージを渡します。相手システムからのメッセージをまだ受信していない場合は，受信するまで dc_mcf_recvsync 関数は待ち続けます。相手システムからメッセージを受信した時点で，dc_mcf_recvsync 関数にメッセージを渡します。

- 同期型のメッセージ送受信

  同期型のメッセージの送信と受信を一つの関数でする形態です。MHP は，dc_mcf_sendrecv 関数【CBLDCMCF('SENDRECV')】で MCF にメッセージの送信要求をします。MCF はメッセージを出力キューに書き込んで，相手システムへ送信します。送信処理が完了したあとも，dc_mcf_sendrecv 関数はリターンしないで，引き続き受信処理をします。受信までの処理が完了した時点で dc_mcf_sendrecv 関数はリターンします。

(2) 同期型のメッセージ処理の時間監視

同期型のメッセージ処理で，無制限に応答を待ち続けるのを避けるため，監視時間を設定できます。この監視時間は，関数の引数 watchtime に設定します。0 を設定した場合は，MCF マネジャ定義の UAP 共通定義で指定した同期送受信監視時間が仮定されます。UAP 共通定義の監視時間に 0 が指定されていた場合は，無制限に応答を待ち続けます。

同期型のメッセージの処理時間は，トランザクションブランチの限界経過時間に含めるか含めないかを選択できます。この指定はユーザサービス定義，ユーザサービスデフォルト定義，トランザクションサービス定義の trn_expiration_time_suspend で指定します。なお，非トランザクション属性の MHP の限界経過時間に含めることはできません。trn_expiration_time_suspend に指定する値とトランザクションの時間監視の詳細については，マニュアル「OpenTP1 システム定義」を参照してください。

注意事項

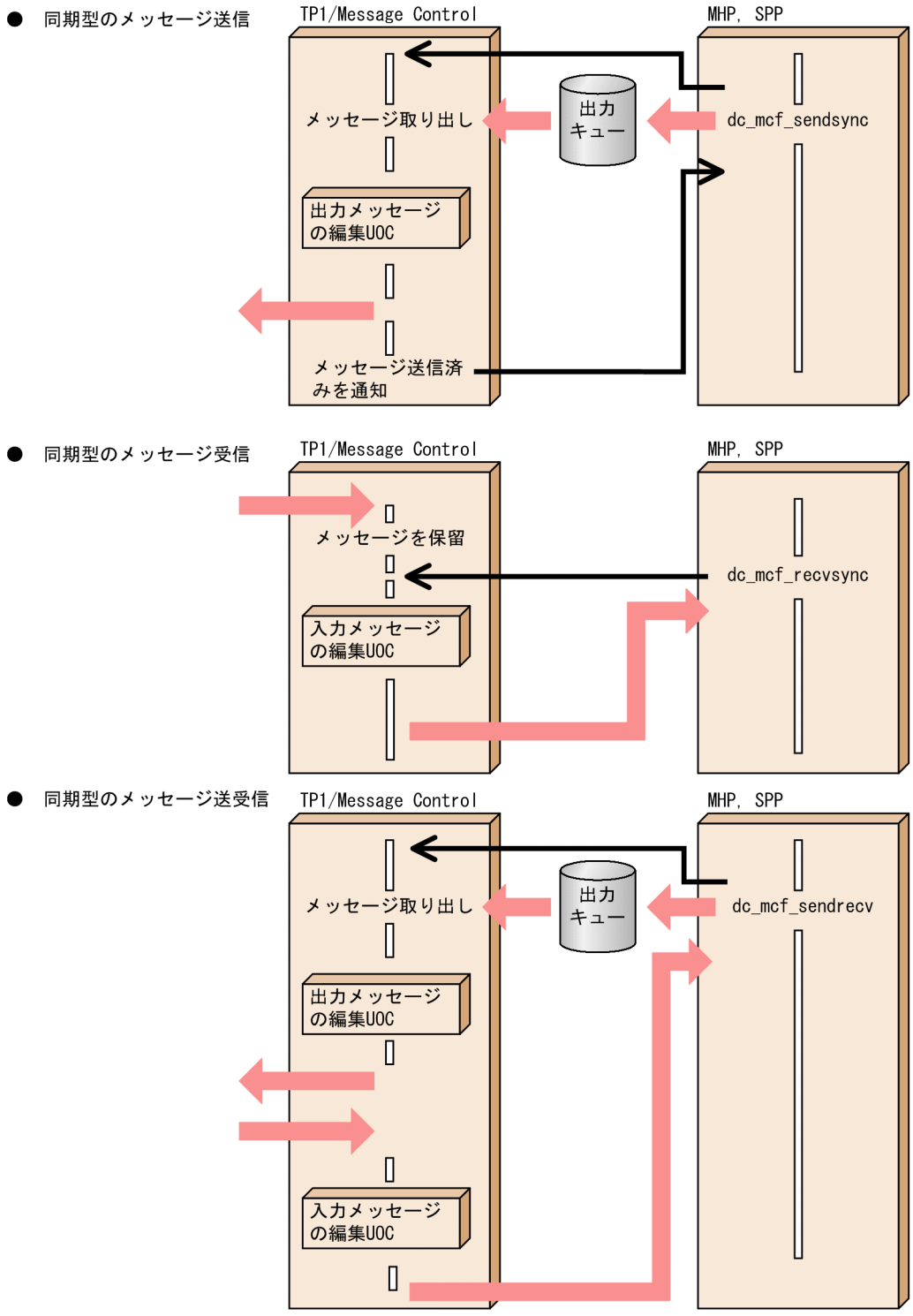
監視時間の精度は秒単位です。また，タイマ定義（mcfttim -t）の btim オペランドで指定する時間監視間隔でタイムアウトが発生したかどうかを監視しています。このため，設定した監視時間と実際にタイムアウトを検出する時間には秒単位の誤差が生じます。そのため，タイミングによっては，設定した監視時間よりも短い時間でタイムアウトすることがあります。監視時間が小さくなるほど，誤差の影響を受けやすくなりますので，監視時間は 3（単位：秒）以上の値の設定を推奨します。

(3) 同期型のメッセージ処理とロールバック

MHP がロールバックされた場合，同期型のメッセージは廃棄されません。ただし，dc_mcf_sendsync 関数，または dc_mcf_sendrecv 関数で複数セグメントを送信した場合に，最終セグメント（EMI）を指定しないで return()を呼び出したときは，メッセージは廃棄されます。

同期型のメッセージ処理を次の図に示します。

図 3-10　同期型のメッセージ処理



3.6.6　継続問い合わせ応答の処理

問い合わせ応答の処理を連続させることで，端末と UAP が交互にメッセージをやり取りする形態です。継続問い合わせ応答は，アプリケーションの型が継続問い合わせ応答型（cont 型）の MHP でだけできます。

(1) 継続問い合わせ応答の処理概要

継続問い合わせ応答をするMHPでは，dc_mcf_receive関数を呼び出して，端末からのメッセージを受信します。処理後dc_mcf_reply関数で応答を返します。応答を返す場合，継続して処理するMHPを変更するときは，dc_mcf_reply関数に変更後のアプリケーション名を設定します。設定しないと，前回と同じMHPが起動されます。

さらに，継続問い合わせ応答処理中のMHPから，dc_mcf_execap関数でアプリケーション起動できます。ただし，**即時起動**だけできます。この場合，dc_mcf_execap関数で起動できるcont型のMHPは一つだけです。また，cont型のアプリケーションを起動したMHPでは，継続応答権が移動しているので，dc_mcf_reply関数は呼び出せません。また，dc_mcf_contend関数も呼び出せません。

継続問い合わせ応答の処理中でも，端末への一方送信メッセージ（dc_mcf_send関数）は送信できます。

(2) 一時記憶データへのアクセス

継続問い合わせ応答では，続けて起動するMHPへ処理を引き継ぐ情報として，**一時記憶データ**を使えます。一時記憶データは論理端末対応に使えるので，複数の論理端末で，同時に一つのMHPを使って継続問い合わせ応答もできます。

一時記憶データ用の領域として更新用領域と回復用領域を共有メモリに確保します。一時記憶データ格納用領域の長さは，MHPごとに，MCFアプリケーション定義で指定します。

一時記憶データは，継続問い合わせ応答の場合にだけ使えます。そのほかのメッセージ通信形態では使えません。

(a) 一時記憶データの受け取り

MHPから一時記憶データを使う場合は，dc_mcf_tempget関数【CBLDCMCF('TEMPGET ')】を呼び出します。一時記憶データ格納用領域が初期状態である場合や，一時記憶データがないときは，MCFアプリケーション属性定義のtempsizeで指定した長さの$(00)_{16}$があるものと見なして，dc_mcf_tempget関数が実行されます。

dc_mcf_tempget関数に指定した受け取り領域長が，一時記憶データ長よりも小さい場合は，指定した長さ分だけ受け取って，超えた部分は切り捨てます。dc_mcf_tempget関数に指定した受け取り領域長が，一時記憶データ長よりも大きい場合は，一時記憶データだけを，受け取り領域に格納します。

(b) 一時記憶データの更新

一時記憶データを更新する場合は，dc_mcf_tempput関数【CBLDCMCF('TEMPPUT ')】を呼び出します。一時記憶データ格納用領域を更新すると，データそのものが入れ替わります。更新データ長は，MCFアプリケーション定義で指定した値を超える値は設定できません。

dc_mcf_tempput関数を呼び出す前には，dc_mcf_tempget関数を必ず呼び出しておきます。呼び出していない場合は，dc_mcf_tempput関数はエラーリターンします。

(3) 継続問い合わせ応答の終了

継続問い合わせ応答は，次のどれかの事象が実行されると終了します。継続問い合わせ応答処理が終了すると，それまで使っていた一時記憶データ格納用領域は削除されます。

- MHP から dc_mcf_contend 関数【CBLDCMCF('CONTEND ')】の呼び出し
- mcftendct コマンドで論理端末名称を指定して強制終了
  mcftendct コマンドは，継続問い合わせ応答の処理でない UAP からも，dc_adm_call_command 関数で実行できます。mcftendct コマンドについては，マニュアル「OpenTP1 運用と操作」を参照してください。
- UAP の異常終了

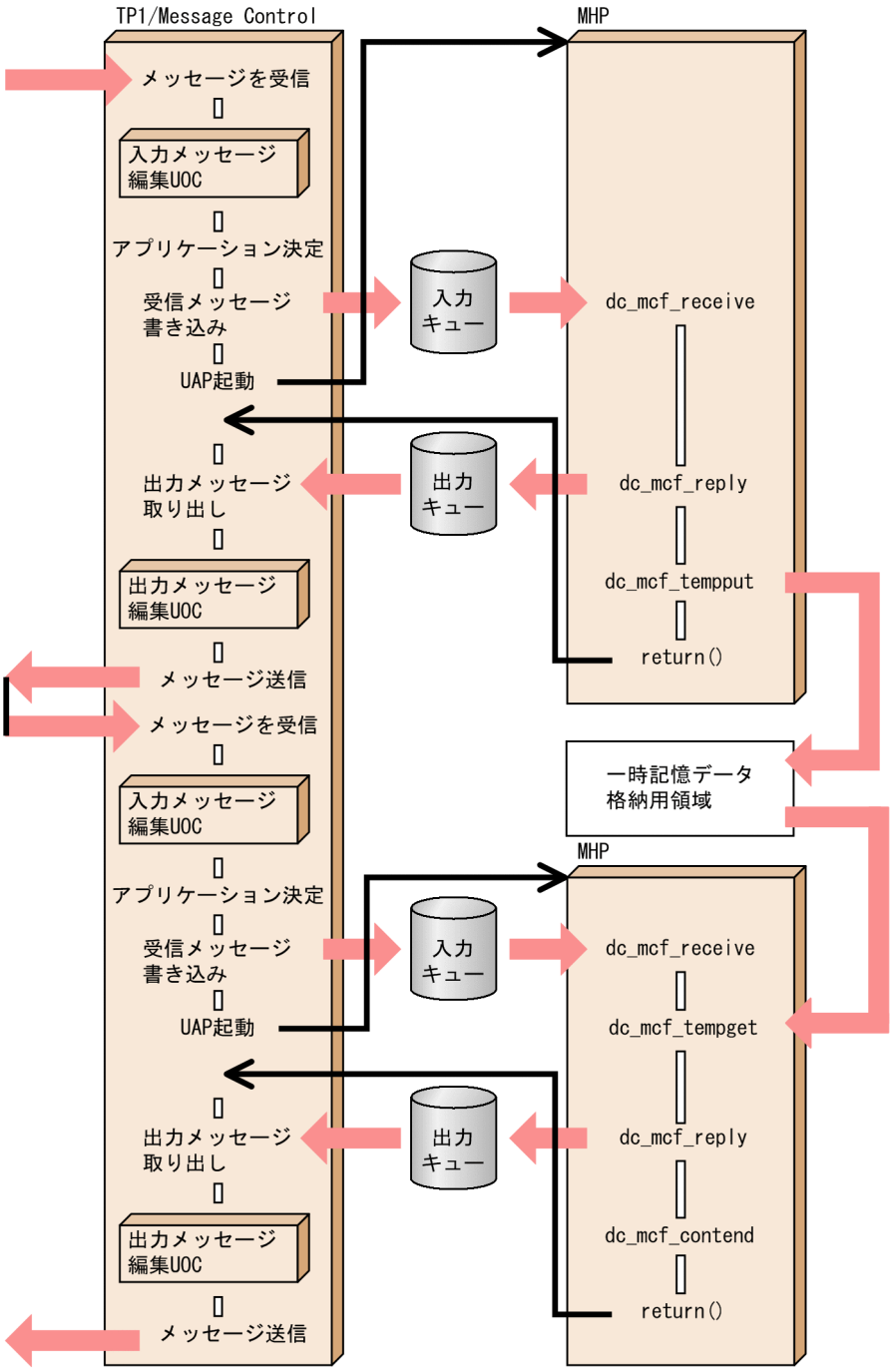
継続問い合わせ応答処理をした MHP で，dc_mcf_reply 関数を呼び出さないで終了しても，異常終了します。

(4) UAP の異常終了によるエラーイベント発生時の処置

継続問い合わせ応答処理中に UAP が異常終了した場合は，ERREVT3 が通知されます。この ERREVT3 を処理する MCF イベント処理用 MHP で，次に起動するアプリケーション名を設定した dc_mcf_reply 関数を呼び出していれば，継続問い合わせ応答の処理を続けられます。呼び出していない場合は，継続問い合わせ応答の処理は終了します。

継続問い合わせ応答の概要を次の図に示します。

図 3-11　継続問い合わせ応答の概要



3.6.7　メッセージの再送

送信したメッセージを，再び送信できます。メッセージは dc_mcf_resend 関数【CBLDCMCF('RESEND')】で再送します。再送するメッセージは，以前に送信したメッセージとは別の，新しいメッセージとして扱います。次のような場合に，メッセージを再送します。

- プリンタに送信したメッセージでの印字が鮮明でない，または文字が破壊されている場合
- 同じ帳票を複数枚必要な場合

- 表示していた画面が消えてしまった場合

(1)　再送できるメッセージの条件

再送の対象にできるのは，次のすべての条件を満たしているメッセージです。

- メッセージ送信時に出力通番を付けていること
- 送信メッセージの割り当て先にディスクキューを指定する論理端末（mcftalcle -k quekind に disk を指定）を使用していること
- 相手システムにメッセージが正常に送信されるなどにより，送信済みになっていること※1
- 送信済みのメッセージがディスクキューに保持されていること※2

注※1

UAP からメッセージを送信したあと，出力キューに滞留したままのメッセージは再送の対象にはなりません。また，-d オプションを省略した mcftdlqle コマンドの入力，または dc_mcf_tdlqle 関数で削除したメッセージは送信済みのメッセージと見なします。一方，-d オプションを指定した mcftdlqle コマンド，または mcftspqle コマンドで削除したメッセージは，送信済みのメッセージと見なしません。

注※2

ディスクキューに保持できるメッセージ数については，「(3)　システムサービス定義との関連」を参照してください。

メッセージキュー（ディスクキュー）内に対象のメッセージがない場合，dc_mcf_resend 関数はエラーリターンします。

(2)　再送対象メッセージの指定内容

どのメッセージを再送するかは，送信済みメッセージに設定してあった，次に示す情報で選択します。

- **出力先の論理端末名称**
  出力先の論理端末名称を指定することで，取り出すメッセージがある出力キューを決定します。
- **メッセージ出力通番**
  出力通番は，次のどちらかで設定します。メッセージを再送するときは，新しく出力通番を付け直せます。
  1. 再送対象メッセージの出力通番を設定
  2. 送信済みメッセージのうち，最終出力通番のメッセージを再送することを設定
- **メッセージ種別（一般分岐，または優先分岐）**
  メッセージを再送するときには，メッセージ種別を新しく指定し直せます。

(3) システムサービス定義との関連

メッセージキュー（ディスクキュー）内に保持する送信済みメッセージ数はメッセージキューサービス定義の quegrp コマンドの-m オプションで指定します。

論理端末ごとにこのオプションで指定したメッセージ数をメッセージキューに保持することができます。

(4) ネットワークコミュニケーション定義との関連

メッセージを再送するとき，MCF マネジャ定義の UAP 共通定義（mcfmuap）の-e オプションで指定した最大セグメント長分の領域だけ，作業領域として使います。再送するメッセージセグメント長が，この作業領域よりも大きい場合は，dc_mcf_resend 関数はエラーリターンします。このため，UAP 共通定義の-e オプションでは，再送するメッセージの最大長以上の値を設定しておいてください。

また，MCF マネジャ定義の UAP 共通定義（mcfmuap）の-l オプションでの，出力通番に関する指定内容によっては，メッセージキューファイル内に同じ出力通番を持ったメッセージが同時に存在する場合があります。この場合，どのメッセージを再送するか保証できません。

3.7　MCF のトランザクション制御

OpenTP1 では，相手システムから受け取ったメッセージの処理をトランザクションとして処理できます。

ここでは，**メッセージ送受信形態の UAP（MHP）**のトランザクション制御について説明します。クライアント／サーバ形態の UAP（SUP，SPP）のトランザクション制御については，「2.3　トランザクション制御」を参照してください。

3.7.1　MHP のトランザクション制御

MHP は，OpenTP1 のメッセージ受信による MHP の開始から，MHP の終了まで，**必ずトランザクション**になります。つまり，MHP で扱う処理はすべて OpenTP1 でトランザクションと見なして処理します。※

MHP のサービス関数では，トランザクション制御をする関数（dc_trn_begin 関数など，dc_trn で始まる同期点取得の関数）は使えません。さらに，MHP から SPP のサービスを要求する場合，サービスを要求された SPP では，トランザクション制御をする関数は呼び出せません。MHP から SPP のサービスを要求するときは，その SPP の処理でトランザクション制御をする関数を呼び出していないことを確認してください。

注※

MCF の拡張機能として，MHP の処理をトランザクションとしないこともできます。これを**非トランザクション属性の MHP** といいます。非トランザクション属性の MHP については，「3.8.3　非トランザクション属性の MHP」を参照してください。

(1)　トランザクション属性の指定

MHP は，ユーザサービス定義で**トランザクション属性である**（atomic_update=Y）ことを指定しておきます。

(2)　関数を使った同期点取得

MHP の処理の中で，連鎖モードのコミットとして同期点を取得できます。同期点は，**dc_mcf_commit 関数【CBLDCMCF('COMMIT ')】**で取得します。dc_mcf_commit 関数が正常に終了すると，続く MHP の処理は新しいグローバルトランザクションとなります。

MHP から始まるグローバルトランザクションが，複数のトランザクションブランチから構成される場合（MHP から dc_rpc_call 関数で SPP を呼び出しているとき）は，それぞれのトランザクションブランチの処理結果がコミットとならないかぎりコミットとなりません。コミットできない場合は，すべてのトランザクションブランチがロールバックされます。

メッセージを受信する前に，dc_mcf_commit 関数で同期点を取得できません。また，dc_mcf_commit 関数で同期点を取得したあとで，その MHP でメッセージを受信できません。

dc_mcf_commit 関数で同期点取得の対象となるメッセージ処理は，非同期のメッセージとアプリケーションプログラムの起動です。同期型のメッセージ送受信の処理は，同期点取得の対象にはなりません。

dc_mcf_commit 関数は，MCF アプリケーション定義で**非応答型（noans 型）**を指定した MHP からだけ呼び出せます。それ以外の型の MHP で呼び出した場合は，エラーリターンします。また，MHP 以外の UAP では，dc_mcf_commit 関数は呼び出せません。

(3) MHP のロールバック処理

(a) MHP の処理が異常終了した場合

MHP が異常終了またはロールバック※1 した場合は，エラーイベントが通知されます。dc_mcf_receive 関数が先頭セグメントを受け取ったかどうかで，通知されるエラーイベントは異なります。

- 先頭セグメントを受け取る前に異常終了した場合：ERREVT2※2
- 先頭セグメントを受け取ってから異常終了した場合：ERREVT3

注※1

MCF アプリケーション定義（mcfaalcap -g）の recvmsg オペランドに r を指定した場合，または dc_mcf_rollback 関数の action に DCMCFRTRY もしくは DCMCFRRTN を指定した場合は除きます。

注※2

非常駐 MHP の場合，次のような原因で MHP が起動できないときは，ERREVT2 は通知されません。

- 該当するロードモジュールが存在しない。
- RPC インタフェース定義ファイルに記載したエントリポイントに対応したサービス関数がない。

このとき，該当するサービスグループの入力キューのスケジュールを閉塞し，受信メッセージを入力キューに残します。

(b) MHP の処理中にエラーが起こった場合

MHP のトランザクション処理がエラーとなったときは，メッセージを受信する前の状態に戻すため，ロールバック（dc_mcf_rollback 関数【CBLDCMCF('ROLLBACK')】）を MHP で呼び出してください。メッセージを受け取った MHP がロールバックしたときに，再び OpenTP1 で MHP をスケジュールし直すかどうかは，dc_mcf_rollback 関数の引数の指定に従います。

- NORETURN（action に DCMCFNRTN）を設定した場合

  ロールバックしたあとで，MHP に制御を戻しません。該当の MHP は異常終了して，ERREVT3 が通知されます。

- RETURN（action に DCMCFRRTN）を設定した場合

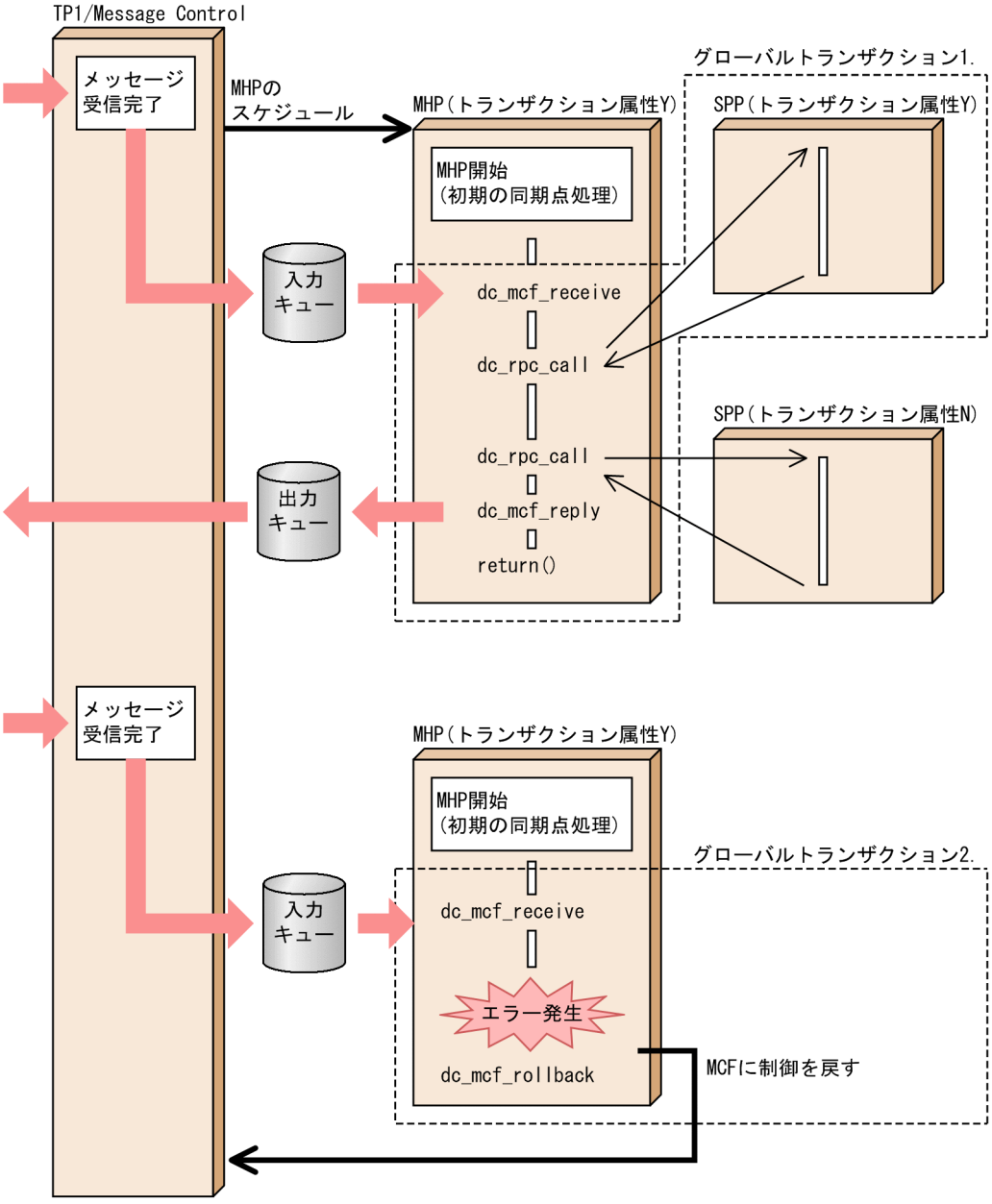
  正常にロールバックできた場合，dc_mcf_rollback 関数が正常に終了します。そのあとで，MHP で任意の処理を続けられます。ロールバックしたあとは新しい別のトランザクションとなります。

- RETRY（action に DCMCFRTRY）を設定した場合

  dc_mcf_rollback 関数がリターンしないで，いったん MHP のプロセスを終了します。ロールバックしたあとで MHP をスケジュールし直します。この値を設定する dc_mcf_rollback 関数を使う場合は，そのノードに**アプリケーション起動プロセス**が必要になります。

メッセージ送受信形態の処理とトランザクションの関係を次の図に示します。

図 3-12　メッセージ送受信形態の処理とトランザクションの関係



（説明）

1. メッセージを受信した時点で，MHP から始まる処理はグローバルトランザクションになります。
2. MHP のトランザクション処理でエラーが起こった場合は，ロールバック（部分回復）してから，MCF に制御を渡します。リターン指定（action に DCMCFRRTN を設定）の dc_mcf_rollback 関数を呼び出すと，新しいトランザクションの処理もできます。

(4) メッセージ送受信関数がエラーリターンしても MHP がコミットとなる場合

MHP の処理で，メッセージ送受信機能の関数がエラーリターンしたあとで，リターンで処理を終了させた場合，そのトランザクション自体はコミットすることがあります。このような MHP の処理の中でリソースマネジャ（RM）にアクセス（DAM，TAM）をしていれば，このアクセスの処理はコミットになります。アクセスの処理もロールバックとしたい場合は，エラーリターン後に dc_mcf_rollback 関数を呼び出す処理を作成しておくか，abort を呼び出しておいてください。

(5) MHP からトランザクションを開始する関数を呼び出す場合

MHP でも，サービス関数の処理範囲以外（メイン関数の処理範囲）ならば，トランザクションの開始（dc_trn_begin 関数）を使えます。トランザクションの開始と同期点取得は，メイン関数の dc_rpc_open 関数から dc_mcf_mainloop 関数の間，または dc_mcf_mainloop 関数から dc_rpc_close 関数の間で呼び出せます。

MHP のメイン関数で dc_trn_begin 関数を呼び出したときは，そのメイン関数で必ず非連鎖モードのコミット（dc_trn_unchained_commit 関数）を呼び出して同期点を取得してください。

3.8 MCF の拡張機能

MCF には，メッセージ送受信以外にも次に示す機能があります。

- アプリケーションプログラムの起動
- コマンドを使った MHP の起動
- 非トランザクション属性の MHP
- ユーザタイマ監視機能による時間監視

3.8.1 アプリケーションプログラムの起動

MHP または SPP から，MHP を起動できます。アプリケーションプログラムを起動する関数（dc_mcf_execap 関数【CBLDCMCF('EXECAP ')】）に，起動させたい MHP のアプリケーション名と，引き渡すメッセージのセグメントを設定します。

(1) アプリケーションプログラムを起動するときに使用する MCF のプロセス

アプリケーション起動機能（dc_mcf_execap 関数）を使う場合，メッセージ送受信の関数（dc_mcf_receive 関数，dc_mcf_send 関数など）とは別の MCF のプロセスを使います。メッセージ送受信で使う MCF のプロセスを**MCF 通信プロセス**，dc_mcf_execap 関数で使う MCF のプロセスを**アプリケーション起動プロセス**といいます。アプリケーション起動プロセスは，通信プロトコルには依存しません。

(2) アプリケーションプログラムを起動する方法

dc_mcf_execap 関数でアプリケーションプログラムを起動できるのは，MHP と SPP です。
dc_mcf_execap 関数を呼び出すと，MHP を起動できます。

dc_mcf_execap 関数で送信したセグメントは，MHP で呼び出す dc_mcf_receive 関数で受け取ります。起動できるのは，dc_mcf_execap 関数を呼び出した UAP と同じノードにある MHP だけです。他ノードの MHP は，dc_mcf_execap 関数で起動できません。※

注※

TP1/NET/OSAS-NIF を使用して通信する場合は，dc_mcf_execap 関数でメッセージ送受信をするので，この限りではありません。

(3) 起動するタイミング

起動させる MHP が実際に起動するのは，次の場合です。

- MHP から dc_mcf_execap 関数を呼び出した場合

  MHP のトランザクションがコミット（MHP が正常に終了，または dc_mcf_commit 関数が正常に終了）してから，起動します。

- SPP から dc_mcf_execap 関数を呼び出した場合

  トランザクションがコミットとなったときに，起動します。

  SPP から dc_mcf_execap 関数を呼び出す場合は，SPP がトランザクションとして処理していることと，その SPP のメイン関数で dc_mcf_open 関数を呼び出していることが前提です。

(4) アプリケーションプログラムの起動の種類

MHP を起動する方法は，次の 2 種類があります。

(a) 即時起動

dc_mcf_execap 関数を呼び出した UAP の処理がコミットとなってから，すぐに起動します。

(b) タイマ起動

dc_mcf_execap 関数を呼び出した直後から，設定した時間に起動します。タイマ起動には，次の 2 とおりの指定があります。

- 経過時間指定のタイマ起動

  dc_mcf_execap 関数を呼び出した直後から，設定した秒数が過ぎてから起動します。設定した秒数が過ぎても dc_mcf_execap 関数を呼び出した UAP の処理がコミットしていない場合は，コミットを待ってからすぐに起動します。

  注意事項

  時間監視の精度は秒単位です。また，タイマ定義（mcfttim -t）の btim オペランドで指定する時間監視間隔で起動するかどうかを監視しています。このため，設定した経過時間と実際に起動する時間には秒単位の誤差が生じます。そのため，タイミングによっては，設定した監視時間よりも短い時間で起動することがあります。監視時間が小さくなるほど，誤差の影響を受けやすくなりますので，監視時間は 3（単位：秒）以上の値の設定を推奨します。

- 時刻指定のタイマ起動

  dc_mcf_execap 関数を呼び出したあと，設定した時刻になったときに起動します。dc_mcf_execap 関数を呼び出した時刻が，関数に設定した時刻を過ぎていた場合に，その場で即時に起動するか，翌日の指定時刻まで持ち越すかを指定します。この指定は，MCF マネジャ定義 UAP 共通定義に指定します。

  アプリケーション起動機能を使用する場合，起動要求を行ってから UAP の開始時刻までの間に夏時間と標準時間の移行が生じたときは，起動要求した時間帯の時刻で UAP が起動されます。

  注意事項

  時間監視の精度は秒単位です。また，タイマ定義（mcfttim -t）の btim オペランドで指定する時間監視間隔で起動するかどうかを監視しています。このため，設定した時刻と実際に起動する時刻には秒単位の誤差が生じます。

(5) アプリケーションプログラムの起動前に障害が起こった場合のエラーイベント

dc_mcf_execap 関数を呼び出して，MHP を起動するまでに障害が起こった場合は，次の MCF イベントが通知されます。

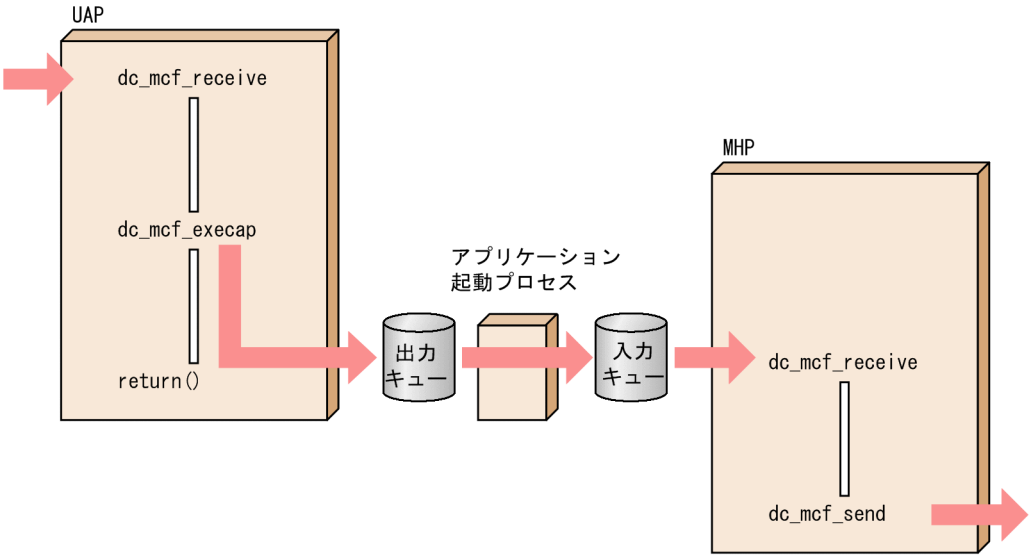
- 即時起動の場合：ERREVT2
- タイマ起動の場合：ERREVT4

MCF イベントについては，「3.10　MCF イベント」を参照してください。

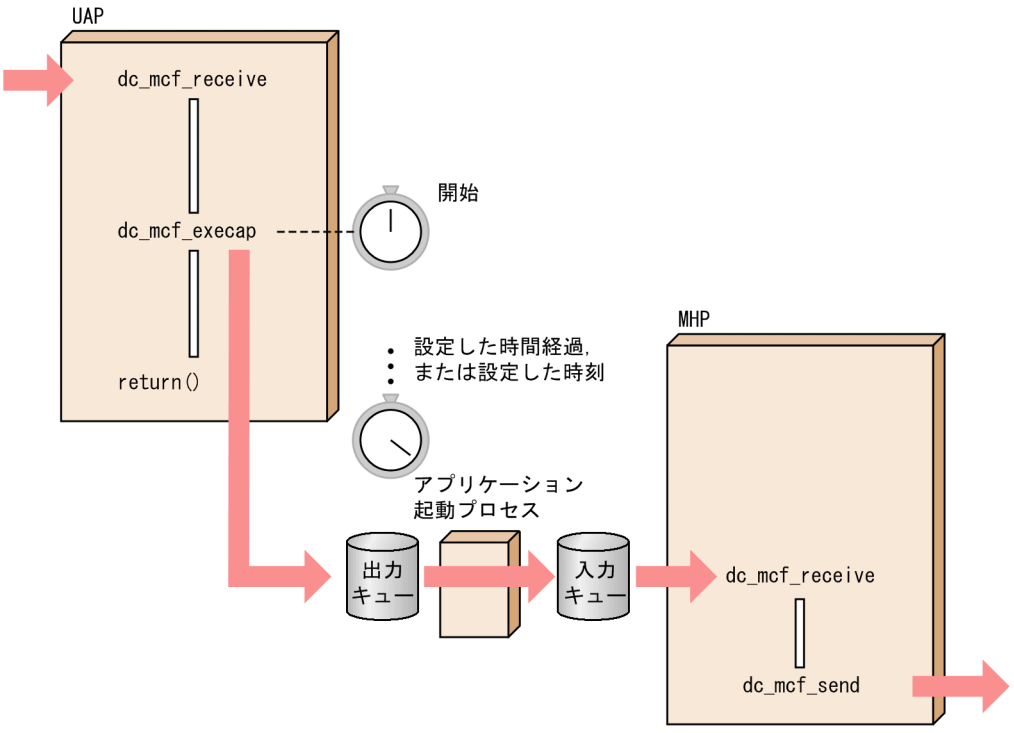
アプリケーションプログラムの起動を次の図に示します。

図 3-13　アプリケーションプログラムの起動

● 即時起動（UAPが正常に終了してから起動）



● タイマ起動（dc_mcf_execap関数を呼び出した直後から，設定した時間に起動）



(6)　システム定義との関連

(a)　MCF 通信構成定義との関係

dc_mcf_execap 関数を呼び出す UAP があるノードには，通常の実行プロセスのほかに，**アプリケーション起動プロセス**が必要になります。アプリケーション起動プロセスは MCF 通信構成定義で定義します。アプリケーションプログラムを起動させる関数を使う OpenTP1 では，MCF 通信構成定義のアプリケーション起動定義を作成しておいてください。

アプリケーション起動定義では，アプリケーション起動環境定義（mcftpsvr）で内部通信路を定義し，アプリケーション起動用論理端末定義（mcftalcle）で論理端末を定義します。

内部通信路や論理端末は複数のアプリケーションで共用できるので，1 対 1 で対応づける必要はありません。

(b) MCF アプリケーション定義との関係

MCF アプリケーション定義のアプリケーション属性定義（mcfaalcap）の type オペランドに指定したアプリケーションの型によって，アプリケーション起動の使い方が決まります。

- 応答型（ans 型）のアプリケーションを起動させる場合

  応答メッセージを送信できる MHP は，応答型（ans 型）を指定した場合だけです。問い合わせメッセージを受信した MHP から ans 型のアプリケーションを起動した場合は，応答権が移動します。そのため，ans 型のアプリケーションは一度だけしか起動できません。さらに，いったん ans 型のアプリケーションを起動させた MHP からは，応答メッセージは送信できません。また，応答メッセージをすでに送信した MHP からは，ans 型のアプリケーションを起動できません。

  SPP から ans 型のアプリケーションを起動できません。

- 非応答型（noans 型）のアプリケーションを起動させる場合

  noans 型のアプリケーションは，一つのトランザクションから複数回起動させることができます。

- 継続問い合わせ応答型（cont 型）のアプリケーションを起動させる場合

  cont 型のアプリケーションを起動できるのは，継続問い合わせ応答処理中の MHP だけです。この場合，即時起動だけで，タイマ起動はできません。継続問い合わせ応答処理中の MHP から dc_mcf_execap 関数を呼び出す場合，アプリケーション起動できる cont 型のアプリケーションは一つだけです。また，cont 型のアプリケーションを起動した MHP では，継続応答権が移動しているので，dc_mcf_reply 関数は呼び出せません。また，dc_mcf_contend 関数も呼び出せません。

  SPP から cont 型のアプリケーションを起動できません。

(c) MCF マネジャ定義との関係

アプリケーションを起動する順序を UAP からの関数の呼び出し順にするか，トランザクションのコミット順にするかは，MCF マネジャ定義の UAP 共通定義（mcfmuap -c）の order オペランドで指定します。

トランザクションのコミットに時間が掛かった場合，同じ論理端末を使用するほかのアプリケーションの起動を待ち合わせるため，起動順序は，トランザクションのコミット順（commit を指定）を推奨します。

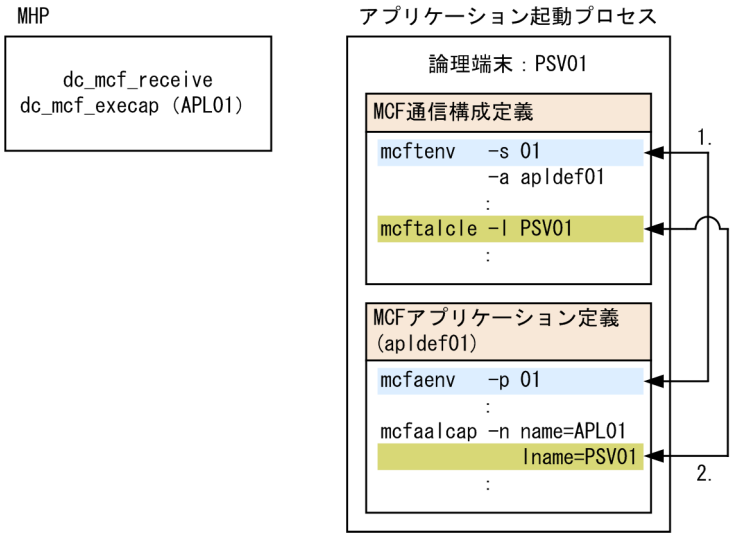
(d) UAP とアプリケーション起動プロセスとの関連づけ

dc_mcf_execap 関数を呼び出す UAP とアプリケーション起動プロセス間で定義する内容を関連づける必要があります。

UAP とアプリケーション起動プロセスとの関連づけを以降の図に示します。

- MHP から非応答型のアプリケーションを起動させる場合

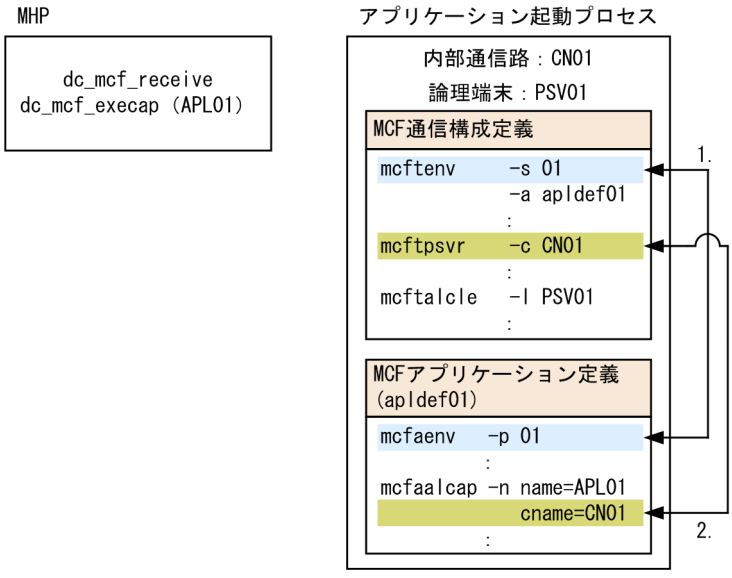
図 3-14　MHP から非応答型のアプリケーションを起動する場合の関連づけ



1. アプリケーション起動プロセスのアプリケーション環境定義のアプリケーション起動識別子（mcfaenv -p）に，MCF 環境定義のアプリケーション起動識別子（mcftenv -s）を指定します。
2. アプリケーション起動プロセスのアプリケーション属性定義の論理端末名称（mcfaalcap -n lname）に，アプリケーション起動用論理端末定義の論理端末名称（mcftalcle -l）を指定します。

- MHP から応答型または継続問い合わせ応答型のアプリケーションを起動させる場合

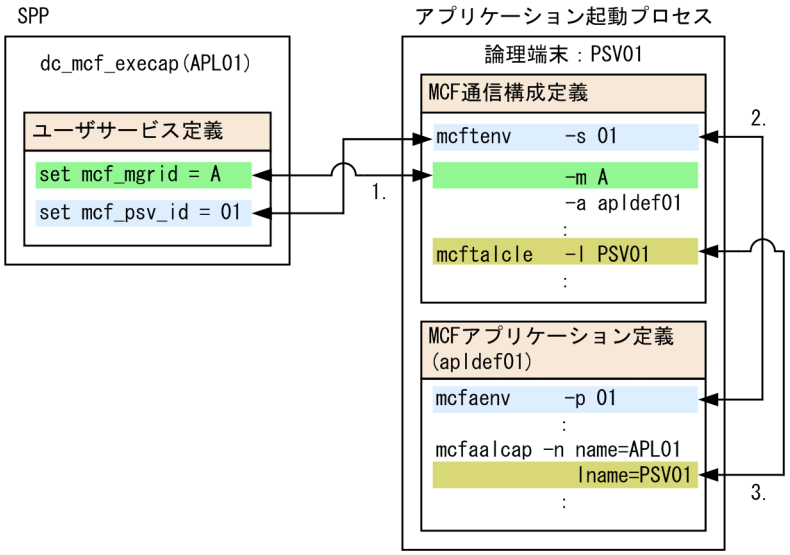
図 3-15　MHP から応答型または継続問い合わせ応答型のアプリケーションを起動する場合の関連づけ



1. アプリケーション起動プロセスのアプリケーション環境定義のアプリケーション起動識別子（mcfaenv -p）に，MCF 環境定義のアプリケーション起動識別子（mcftenv -s）を指定します。
2. アプリケーション起動プロセスのアプリケーション属性定義の内部通信路名（mcfaalcap -n cname）に，アプリケーション起動環境定義の内部通信路名（mcftpsvr -c）を指定します。

- SPP から非応答型のアプリケーションを起動させる場合

図 3-16　SPP から非応答型のアプリケーションを起動する場合の関連づけ



1. ユーザサービス定義の MCF マネジャ識別子（mcf_mgrid オペランド）に，MCF 環境定義の MCF マネジャプロセス識別子（mcftenv -m）を指定します。
2. アプリケーション起動プロセスのアプリケーション環境定義のアプリケーション起動識別子（mcfaenv -p）とユーザサービス定義のアプリケーション起動識別子（mcf_psv_id オペランド）に，MCF 環境定義のアプリケーション起動識別子（mcftenv -s）を指定します。
3. アプリケーション起動プロセスのアプリケーション属性定義の論理端末名称（mcfaalcap -n lname）に，アプリケーション起動用論理端末定義の論理端末名称（mcftalcle -l）を指定します。

(7)　起動させる MHP に渡す，入力元論理端末名称

MHP から dc_mcf_execap 関数で MHP を起動する場合，起動された MHP で受け取るメッセージ入力元の論理端末名称は，最初に受信したメッセージ中の名称になります。さらに，その MHP から dc_mcf_execap 関数を呼び出した場合も，受け取るメッセージ入力元の論理端末名称は，最初にメッセージを受信したときの名称が引き渡されます。

SPP から dc_mcf_execap 関数で MHP を起動する場合，起動された MHP で受け取るメッセージ入力元の論理端末名称は「*」となります。さらに，その MHP から dc_mcf_execap 関数を呼び出した場合も，受け取るメッセージ入力元の論理端末名称は，「*」となります。

アプリケーションプログラムの起動形態と type オペランドの指定を以降の図で示します。

図 3-17　一方送信メッセージを受信した MHP からの起動

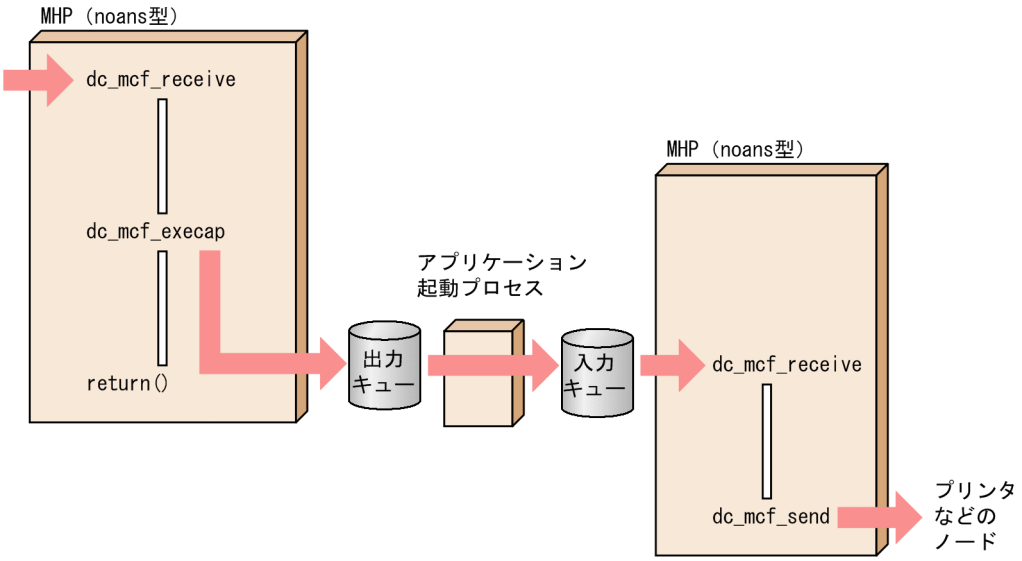
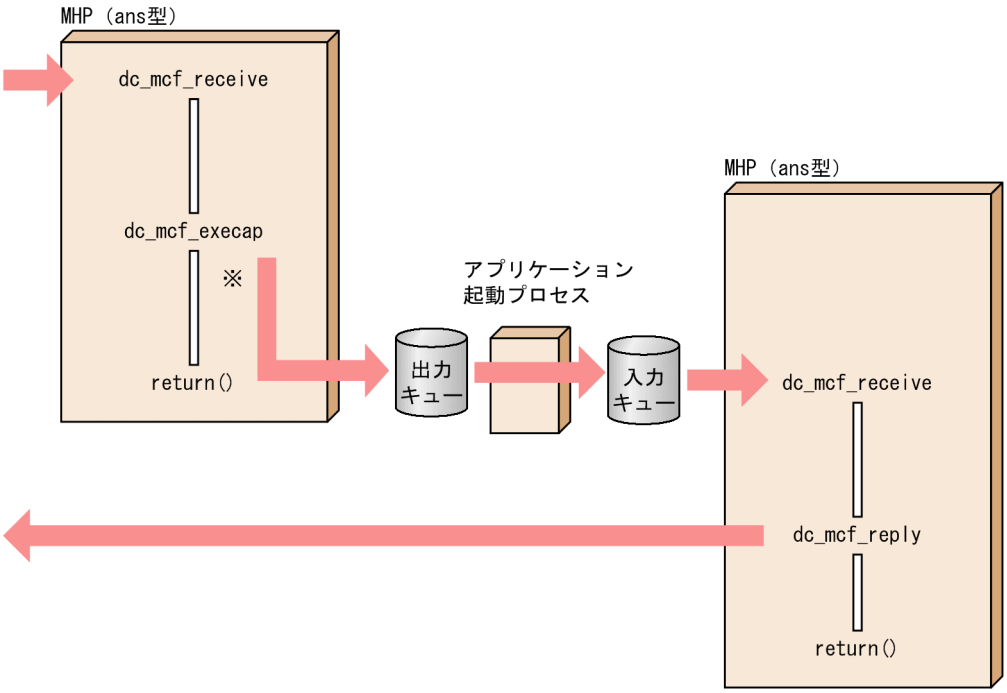


図 3-18　問い合わせ応答メッセージを受信した MHP からの起動



注※　ans型のMHPの起動を要求したあとで，dc_mcf_reply関数を呼び出すと，エラーリターンします。

図 3-19　問い合わせ応答メッセージの処理の MHP から，一方送信メッセージを送信する MHP の起動

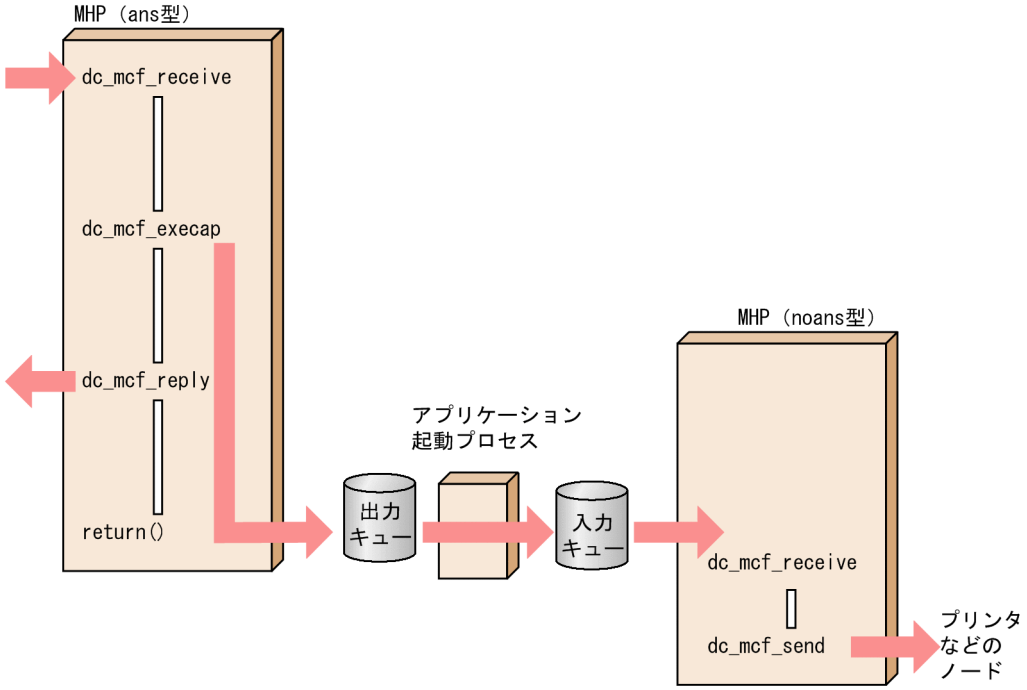
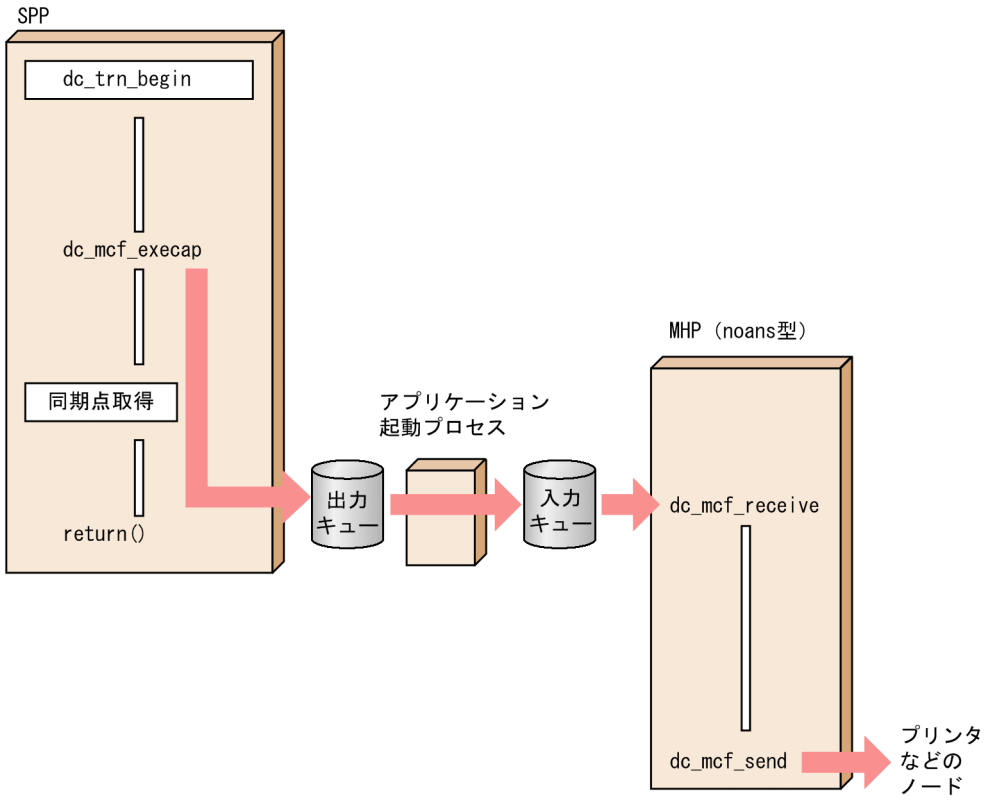


図 3-20　トランザクション処理の SPP からの起動



(8) TP1/Message Control（MCF）の再開始（リラン）時のタイマ起動の扱い

タイマ起動の時間待ちの間に障害が起こって，OpenTP1 を再開始（リラン）した場合の扱いについて説明します。再開始（リラン）後にタイマ起動を引き継げるのは，ディスクキューを使っている場合だけです。再開始（リラン）した場合にタイマ起動をする dc_mcf_execap 関数の扱いは次のとおりです。

(a) タイマ起動の引き継ぎの定義

MCF 通信構成定義 mcftpsvr 定義コマンドの-o オプションに reruntm=yes と指定した場合は，再開始（リラン）する前のタイマ起動メッセージを引き継ぎます。dc_mcf_execap 関数に設定した時間を過ぎていた場合は，即時起動で引き継ぎます。時間を過ぎていない場合は，時間が来るまで待ってから起動します。

reruntm=no と指定した場合は，再開始（リラン）後にはタイマ起動を引き継ぎません。この場合は，もう一度 UAP からタイマ起動の dc_mcf_execap 関数を呼び出してください。

(b) UOC でタイマ起動を引き継ぐ条件の変更

タイマ起動を引き継ぐ場合，UOC でタイマ起動を引き継ぐ条件を変更できます。この UOC を**タイマ起動引き継ぎ決定 UOC** といいます。タイマ起動引き継ぎ決定 UOC を使う場合は，MCF 通信構成定義 mcftpsvr 定義コマンドの-o オプションに reruntm=yes と指定しておいてください。

タイマ起動引き継ぎ決定 UOC については，「3.9.2　タイマ起動引き継ぎ決定 UOC」を参照してください。

3.8.2　コマンドを使った MHP の起動

OpenTP1 のコマンド（**mcfuevt コマンド**）を入力して，MHP を起動できます。メッセージ受信を契機に起動する MHP でも，mcfuevt コマンドで直接 MHP を起動して，他システムへメッセージを送信できるようになります。

起動できる MHP は，**非応答型**（**noans 型**）だけです。mcfuevt コマンドで起動する MHP には，noans 型を指定してください。

(1) コマンドで起動する MHP の定義

mcfuevt コマンドで起動する MHP のアプリケーション名は，**UCMDEVT** とします。MCF アプリケーション定義アプリケーション属性定義 mcfaalcap の-n オプションには，次の値を指定しておきます。

name オペランド：UCMDEVT

kind オペランド：user（または省略）

type オペランド：noans（または省略）

(2)　MHP の起動方法

MHP を起動するときは，**mcfuevt コマンド**を実行します。mcfuevt コマンドの引数には，**MCF 通信プロセス識別子**と **MHP に渡す入力メッセージ**を指定します。

UCMDEVT を定義していない場合に mcfuevt コマンドを実行したときは，mcfuevt コマンドはエラーリターンします。このとき，ERREVT1 は通知されません。

コマンドで起動する MHP は通信プロトコルに依存しないので，mcfuevt コマンドに指定する MCF のプロセスには，**アプリケーション起動プロセス**を指定することをお勧めします。

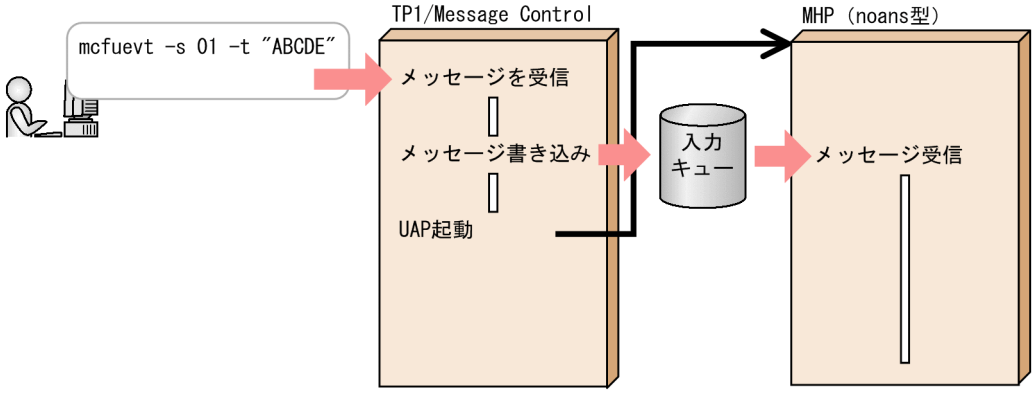
(3)　コマンドで起動した MHP の入力元論理端末名称とコネクション名

コマンドで起動した MHP の入力元論理端末名称は「@UCEVxxx」（xxx は，MCF プロセス識別子）となります。この入力元論理端末名称に対して，UAP からメッセージを送信すると，その関数はエラーリターンします。

コネクション名は「********」となります。

コマンドによる MHP の起動を次の図に示します。

図 3-21　コマンドによる MHP の起動



3.8.3　非トランザクション属性の MHP

トランザクションとして稼働しない MHP（**非トランザクション属性の MHP**）を作成できます。非トランザクション属性の MHP はトランザクションとして処理できませんが，従来の MHP の処理に比べて，処理速度を向上できます。

(1)　トランザクション処理の MHP との違い

トランザクションとして稼働する MHP と同様のメッセージ送受信の関数を使えます。ただし，次の点が異なります。

- メッセージ出力通番は設定できますが，エラー時の回復対象として出力通番を使えません。

- 同期点処理をする関数（dc_mcf_commit 関数，dc_mcf_rollback 関数）は呼び出せません。また，メッセージの再送（dc_mcf_resend 関数）も呼び出せません。このような関数を呼び出した場合はエラーリターンします。

(2) 非トランザクション属性の MHP の定義

(a) 使えるメッセージキュー

非トランザクション属性の MHP では，**メモリキュー**だけ使えます。ディスクキューは使えません。MCF アプリケーション定義アプリケーション属性定義 mcfaalcap の-g オプション quekind オペランドには，メモリキューを指定してください。

(b) MHP のトランザクション属性

非トランザクション属性の MHP には，MCF アプリケーション定義アプリケーション属性定義の trnmode オペランドに nontrn を指定します。ユーザサービス定義の atomic_update の値が Y でも，トランザクションでない処理となります。

(c) 時間監視

非トランザクション属性の MHP の**時間監視**は，MCF アプリケーション定義アプリケーション属性定義 mcfaalcap オペランドの-v オプションで指定します。ここに指定した時間を過ぎると，非トランザクション属性の MHP は異常終了します。この定義に 0 を指定した場合は，時間監視はしません。

非トランザクション属性の MHP から同期型のメッセージ通信関数を呼び出した場合，この処理時間は監視時間に含みません。非トランザクション属性の MHP から SPP のサービスを要求した場合は，SPP の処理時間も監視時間に含みます（この SPP で同期型のメッセージを処理している場合，その時間も監視時間に含みます）。

注意事項

サービス関数開始から終了までの実行監視時間の精度は秒単位です。そのため，タイミングによっては，指定した監視時間よりも短い時間でプロセスを強制停止することがあります。サービス関数開始から終了までの実行監視時間が小さくなるほど，誤差の影響を受けやすくなりますので，このオペランドには 3（単位：秒）以上の値の指定を推奨します。

(3) 非トランザクション属性の MHP で障害が起こった場合

MCF アプリケーション定義の指定によって，ERREVT2，または ERREVT3 の MCF イベント処理用 MHP が起動されます。非トランザクション属性の MHP から SPP のサービスを要求している場合，SPP の処理に対しては何もしません。

継続問い合わせ応答形態の処理の場合に，一時記憶データの実更新ができなかったときは，エラーイベントの定義に関係なく，継続問い合わせ応答処理を終了させます。システム内部の障害などで，継続問い合わせ応答処理を終了できない場合は，継続問い合わせ応答の強制終了コマンド（mcftendct -f）の実行を

要求するメッセージログが出力されるので，コマンドを実行して継続問い合わせ応答処理を終了させてください。

3.8.4 ユーザタイマ監視機能による時間監視

MHP または SPP から関数で時間監視を設定したり，その設定を取り消したりできます。この機能を**ユーザタイマ監視機能**といいます。これによって，ユーザで任意の時間監視ができます。ユーザタイマ監視機能を使用するには，MCF 通信構成定義 mcfttim の-p オプションに usertime=yes を指定する必要があります。

ユーザタイマ監視を設定するには dc_mcf_timer_set 関数【CBLDCMCF('TIMERSET')】を呼び出し，ユーザタイマ監視を取り消すには dc_mcf_timer_cancel 関数【CBLDCMCF('TIMERCAN')】を呼び出します。ユーザタイマ監視の設定および取り消しは，トランザクションに関係なく，関数呼び出し時点で処理されます。

タイムアウトが発生したかどうかは，MCF が一定の時間監視間隔で行います。時間監視間隔は MCF 通信構成定義 mcfttim の-t オプションの btim オペランドで指定します。

タイムアウトが発生した場合，dc_mcf_timer_set 関数の引数に指定した MHP を起動させます。dc_mcf_timer_set 関数の引数にユーザデータを指定しておくと，タイムアウトが発生した場合に起動させる MHP に，そのデータをメッセージとして渡せます。

なお，mcftlsutm コマンドを使用すると，ユーザタイマ監視の状態を表示できます。mcftlsutm コマンドについては，マニュアル「OpenTP1 運用と操作」を参照してください。

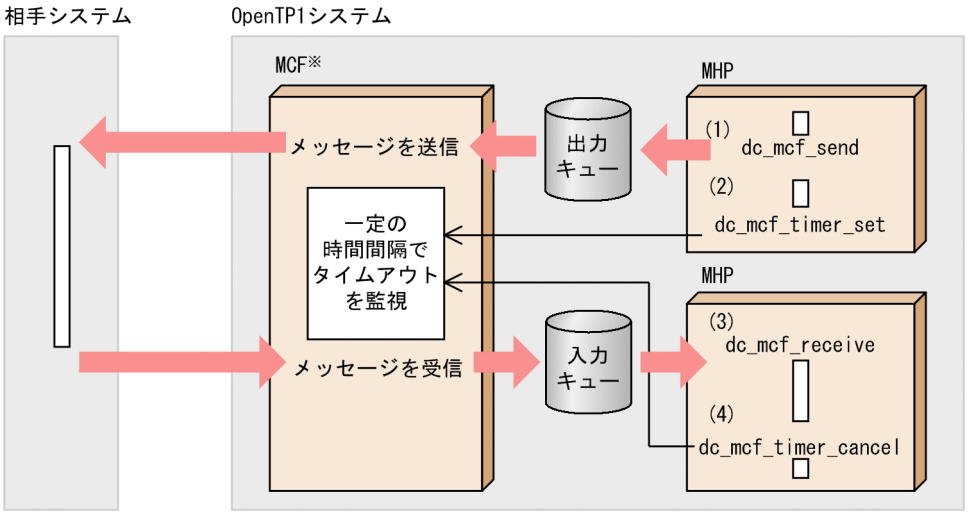
ユーザタイマ監視機能はすべてのプロトコルで使用できます。

(1) 使用例

相手システムからの応答の時間監視を例に，ユーザタイマ監視機能の使用例を次の図に示します。

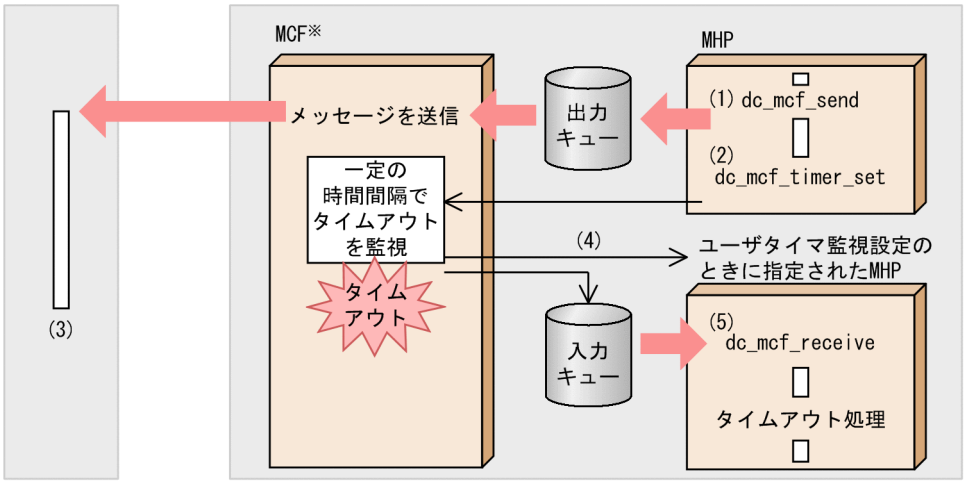
図 3-22　ユーザタイマ監視機能の使用例

●ユーザタイマ監視時間内に応答を受信した場合



(1) 相手システムへメッセージを送信します。
(2) dc_mcf_timer_set関数を呼び出してユーザタイマ監視を設定します。
(3) 相手システムからの応答メッセージを受信します。
(4) 該当するユーザタイマ監視をdc_mcf_timer_cancel関数で取り消します。

●ユーザタイマ監視時間が経過しても応答がない場合



(1) 相手システムへメッセージを送信します。
(2) dc_mcf_timer_set関数を呼び出してユーザタイマ監視を設定します。
(3) 何らかの理由で相手システムから応答メッセージが送信されていません。
(4) MCFはタイムアウトを検出したので，ユーザタイマ監視設定のときに指定されたMHPを起動します。
(5) dc_mcf_timer_set関数の引数にユーザデータを指定した場合，MHPにそのデータをメッセージとして渡します。

注※　MCF：TP1/Message Control，TP1/NET/Library，および通信プロトコル対応の製品の総称

(2)　ユーザタイマ監視機能を使用する場合の注意事項

1. ユーザタイマ監視の設定および取り消しは，関数呼び出し時点で処理されます。そのため，該当するトランザクションがロールバックしても，ユーザタイマ監視の設定および取り消し処理が無効となることはありません。

2. タイムアウト発生時に起動させる MHP は，非応答型（noans 型）の MHP でなくてはなりません。MHP または SPP から dc_mcf_timer_set 関数を呼び出してユーザタイマ監視を設定する場合に，引数に指定した MHP が非応答型でないときは，dc_mcf_timer_set 関数がエラーリターンします。

3. 時間監視の精度は秒単位です。また，タイマ定義（mcfttim -t）の btim オペランドで指定する時間監視間隔でタイムアウトが発生したかどうかを監視しています。このため，設定した監視時間と実際にタイムアウトを検出する時間には秒単位の誤差が生じます。そのため，タイミングによっては，設定した監視時間よりも短い時間で起動することがあります。監視時間が小さくなるほど，誤差の影響を受けやすくなりますので，監視時間は 3（単位：秒）以上の値の設定を推奨します。

4. タイムアウト発生によって MHP が起動される直前に，dc_mcf_timer_cancel 関数が呼び出されると，この関数が「タイムアウト発生済み」でエラーリターンしたあとに，該当する MHP が起動されることがあります。

5. ユーザタイマ監視機能使用時にタイムアウトが頻発すると，通常のメッセージ制御処理の性能に影響します。タイムアウト発生によってアプリケーションを起動させることを，正常時の処理とする使い方はしないでください。

6. ユーザタイマ監視の要求数の最大値を通信構成定義 mcfttim の-p オプションの timereqno オペランドに指定しておく必要があります。このオペランドで指定した値によって，MCF は事前に要求数分の監視用テーブルを静的共用メモリ上に確保します。1 件の設定に対して「約 100 バイト+ユーザデータサイズ」の静的共用メモリが必要です。すべての MCF での静的共用メモリの合計値を，MCF マネジャ定義 mcfmcomn の-p オプション，およびシステム環境定義の static_shmpool_size オペランドの指定値に加算してください。

7. システムダウン時に時間監視中であった場合，再開始（リラン）時に無効となります。ただし，入力キューにディスクキューを使用した場合，タイムアウト発生によって MHP を起動する直前で，システムダウンした場合，再開始後に MHP を起動する可能性があります。したがって，入力キューにはメモリキューを使用することをお勧めします。

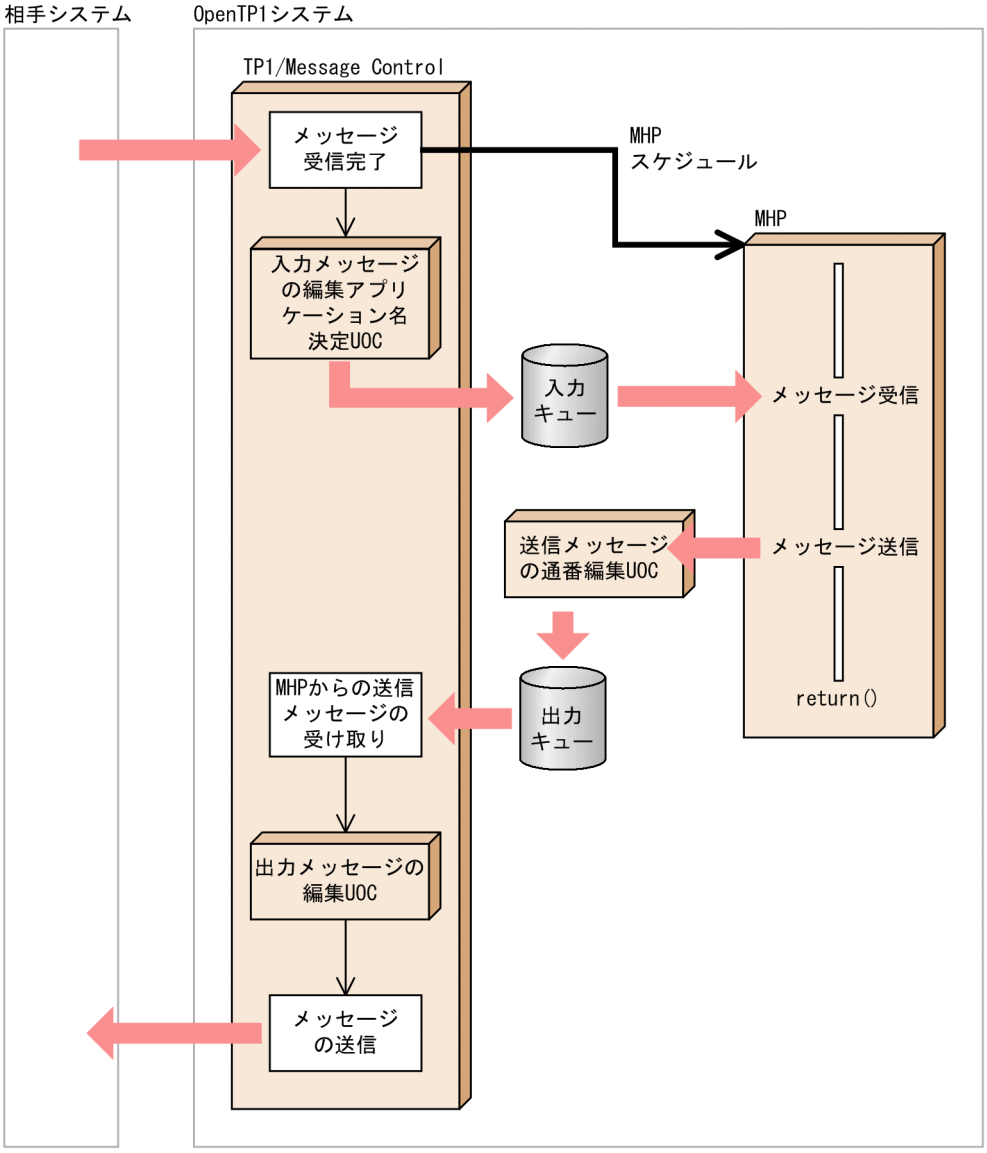
3.9　ユーザオウンコーディング（UOC）

OpenTP1 でメッセージ送受信形態の通信をする場合に，UAP のメッセージ処理を補助するために作成するプログラムを**ユーザオウンコーディング**（UOC　User Own Coding）といいます。UOC は，業務に合わせて任意に作成してください。

UOC のコーディングには，**C 言語**または **C++言語**を使います。C 言語の場合は，**ANSI C** の形式または ANSI 準拠前の **K&R の形式**のどちらかに従ってコーディングします。C++言語の場合は，**C++言語の仕様**でコーディングします。

メッセージ送受信の処理とユーザオウンコーディングの位置を次の図に示します。

図 3-23　メッセージの送受信の処理とユーザオウンコーディングの位置



- **OpenTP1 で使えるユーザオウンコーディング**

  OpenTP1 で使える UOC の一覧を次の表に示します。

表 3-12　OpenTP1 で使えるユーザオウンコーディング

UOC の種類	UOC でできる処理	UOC を使わないときの処理
入力メッセージの編集， アプリケーション名決定 UOC	• 受信したメッセージを編集します。 • メッセージを処理する MHP のアプリケーション名を決定します。	先頭セグメントの，最初の空白までの先頭 8 文字をアプリケーション名とします。
送信メッセージの通番編集 UOC	• 送信するセグメントに出力通番を付けます。	メッセージを編集しないで出力キューに書き込みます。
タイマ起動引き継ぎ決定 UOC	• 再開始（リラン）後のタイマ起動のアプリケーション起動機能の条件を変更できます。	定義の指定によって，引き継ぐかどうかが決まります。
出力メッセージの編集 UOC	• 出力するメッセージを編集します。	出力するメッセージを編集しないで送信します。

この表に示す UOC は，MCF で使う通信プロトコル対応製品によって文法が異なります。また，この表に示す UOC 以外にも，通信プロトコル対応製品で固有の UOC が使える場合があります。UOC の文法については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

この表に示す UOC のうち，**タイマ起動引き継ぎ決定 UOC** は通信プロトコル対応製品に依存しない UOC です。タイマ起動引き継ぎ決定 UOC の文法については，マニュアル「OpenTP1 プログラム作成リファレンス C 言語編」を参照してください。

3.9.1　入力メッセージの編集 UOC，アプリケーション名決定 UOC

入力メッセージの内容を処理する MHP を決めるアプリケーション名を決定する UOC です。この UOC を組み込んだ場合，OpenTP1 で他システムからのメッセージを受け取ると，最初にこの UOC に渡されます。UOC の処理が終了すると，メッセージのデータは入力キューに渡ります。そして，OpenTP1 でスケジュールされた MHP のメッセージの受信をする関数に渡されます。

MCF イベントが通知された場合に，MCF イベント処理用 MHP で回復処理をするときは，入力メッセージの編集とアプリケーション名決定 UOC は経由しません。

入力メッセージの編集とアプリケーション名決定 UOC の形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

(1)　OpenTP1 への組み込み方法

MCF メイン関数（スタート関数　dc_mcf_svstart 関数）に，作成した UOC の関数アドレスを指定します。入力メッセージの編集 UOC の関数アドレスは任意に決められます。UOC のオブジェクトファイルは，MCF メイン関数を翻訳・結合すれば，MHP の実行形式ファイルに結合されて実行できる状態になります。MCF メイン関数については，マニュアル「OpenTP1 運用と操作」を参照してください。

3.9.2　タイマ起動引き継ぎ決定 UOC

タイマ起動の dc_mcf_execap 関数を呼び出したあとで，障害が起こって OpenTP1 を再開始（リラン）した場合，タイマ起動の環境を変更する UOC です。タイマ起動引き継ぎ決定 UOC を作成しておくと，次に示すことができます。

- タイマ起動を引き継ぐ，または取り消す。
- 引き継いだタイマ起動を即時起動とする。
- 起動するアプリケーション名を変更する。

(1)　OpenTP1 への組み込み方法

アプリケーション起動サービス用の MCF メイン関数（dc_mcf_svstart 関数）に作成した UOC の関数アドレスを指定します。関数アドレスは任意です。UOC のオブジェクトファイルは，MCF メイン関数を翻訳・結合すれば，アプリケーション起動サービスの実行形式ファイルに結合されて実行できる状態になります。アプリケーション起動サービス用の MCF メイン関数については，マニュアル「OpenTP1 運用と操作」を参照してください。

3.9.3　送信メッセージの通番編集 UOC

送信メッセージに出力通番を付ける UOC です。MHP からメッセージを送信する関数の指定で起動する UOC です。

送信メッセージの通番編集 UOC は，send_uoc 関数として作成します。ただし，送信メッセージの通番編集 UOC はメッセージを送信する関数の最初のセグメントを送信するときに起動するので，この UOC では第 1 セグメントだけしか編集できません。

送信メッセージの通番編集 UOC の形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

(1)　OpenTP1 への組み込み方法

MHP のメイン関数の中に，dc_mcf_regster 関数として登録します。

3.9.4　出力メッセージの編集 UOC

応答メッセージ，または一方送信メッセージの編集をする UOC です。出力メッセージの編集 UOC は，UAP が送信したメッセージを他システムに実際に送信する前に処理するように位置させます。

出力メッセージの編集 UOC の形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

(1) OpenTP1 への組み込み方法

入力メッセージの編集とアプリケーション名決定 UOC と同じように，MCF メイン関数で呼び出すスタート関数に関数アドレス名を指定します。出力メッセージの編集 UOC の関数アドレスは任意に決められます。MCF メイン関数については，マニュアル「OpenTP1 運用と操作」を参照してください。

3.10 MCF イベント

メッセージ送受信形態の通信をする場合，OpenTP1 の各種システム情報を MHP に知らせるために，MCF からメッセージを出力します。これを**MCF イベント**といいます。メッセージ送受信の処理でエラーや障害が起こった場合，システム内で何が起こっているのかが MCF イベントの内容でわかります。MCF イベントには，エラーや障害発生などの**エラーイベント**と，コネクションの確立・解放などプロトコルに依存する**通信イベント**の 2 種類があります。

MCF イベントの内容を基に障害の対処をする MHP を**MCF イベント処理用 MHP** といいます。この MHP を作成しておくと，独自の障害回復処理などができます。

MCF イベントは入力キューに渡されて，MCF イベント処理用 MHP が起動されます。このとき，入力メッセージの編集とアプリケーション名決定 UOC は経由しません。また，MCF イベントに対して障害が起こったことによって，MCF イベントが起動されることはありません。

MCF イベントの一覧を次の表に示します。次の表に示す MCF イベント以外にも，通信プロトコル対応製品で固有な MCF イベントが通知される場合があります。通信プロトコル対応製品で固有な MCF イベントについては，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

表 3-13 MCF イベントの一覧

MCF イベント名	MCF イベントコード	MCF イベントが通知された原因	MCF イベント処理用 MHP で実行する処理の例
不正アプリケーション名検出通知イベント	ERREVT1	メッセージのアプリケーション名が MCF アプリケーション定義にありません。	該当するアプリケーション名がなかったことを知らせます。 受信したメッセージが問い合わせメッセージの場合は，応答メッセージを送信できます。
メッセージ廃棄通知イベント	ERREVT2	次に示す理由で，MCF で受信した入力キューのメッセージ，またはアプリケーションの即時起動によって入力キューに入力されたメッセージを廃棄しました。 • 入力キューに障害が発生しました。 • 入力メッセージ最大格納数を超過しました。 • 動的共用メモリが不足しました。 • キューファイルが満杯になりました。 • MHP のサービス，サービスグループ，またはアプリケーションが閉塞しています。 • スケジュール閉塞されているサービスグループの入力キューに未処理受信メッセージが残った状態で，OpenTP1 を正常終了または計画停止 A で終了しました。 • MHP のサービスグループ，またはアプリケーションがセキュア状態です。	メッセージを廃棄したことを知らせます。 受信したメッセージが問い合わせメッセージの場合は，応答メッセージを送信できます。

MCF イベント名	MCF イベントコード	MCF イベントが通知された原因	MCF イベント処理用 MHP で実行する処理の例
メッセージ廃棄通知イベント	ERREVT2	• MHP で呼び出す dc_mcf_receive 関数にセグメントを渡す前に，MHP が異常終了しました。 • アプリケーション名に相当する MHP のサービスがありません。 • ユーザサーバ未起動などによって，MHP の起動に失敗しました。 • DBMS の障害などによって，トランザクションの開始に失敗しました。	メッセージを廃棄したことを知らせます。 受信したメッセージが問い合わせメッセージの場合は，応答メッセージを送信できます。
UAP 異常終了通知イベント	ERREVT3	MHP で呼び出す dc_mcf_receive 関数にセグメントを渡したあとで，MHP が異常終了，またはロールバック※しました。	UAP が異常終了，またはロールバックしたことを知らせます。 受信したメッセージが問い合わせメッセージの場合は，応答メッセージを送信できます。
タイマ起動メッセージ廃棄通知イベント	ERREVT4	タイマ起動のアプリケーション起動によって入力キューに入力されたメッセージを ERREVT2 に示す理由で廃棄しました。	メッセージを廃棄したことを知らせます。 受信したメッセージが問い合わせメッセージの場合は，応答メッセージを送信できます。
未処理送信メッセージ廃棄通知イベント	ERREVTA	次に示す理由で，UAP から送信した未処理送信メッセージを廃棄しました。 • MCF の正常終了処理時に，未処理送信メッセージの滞留時間監視の時間切れ（タイムアウト）が起こりました。 • mcftdlqle コマンドまたは dc_mcf_tdlqle 関数で，出力キューが削除されました。 • タイマ起動要求や閉塞されている論理端末の出力キューに未処理送信メッセージが残った状態で，dcstop コマンドが実行されました。	未処理送信メッセージを廃棄したことを知らせます。 受信した未処理送信メッセージは，任意のファイルへ退避します。
送信障害通知イベント	SERREVT	メッセージを送信する途中で，通信プロトコルの障害が起こりました。	通信プロトコルの障害でメッセージを送信できなかったことを知らせます。
送信完了通知イベント	SCMPEVT	相手システムへ，メッセージを正常に送信できました。	相手システムまでメッセージを正常に送信できたことを知らせます。
障害通知イベント	CERREVT（VERREVT）	通信管理プログラムで，コネクション障害，または論理端末障害が起こりました。コネクション確立再試行を定義している場合は，通知されません。	コネクション，または論理端末に障害が起こったことを知らせます。
状態通知イベント	COPNEVT（VOPNEVT）	コネクションが確立しました。	メッセージを送受信できることを知らせます。

MCF イベント名	MCF イベントコード	MCF イベントが通知された原因	MCF イベント処理用 MHP で実行する処理の例
状態通知イベント	CCLSEVT（VCLSEVT）	コネクションが正常に解放されました。	メッセージを送受信できないことを知らせます。

注

ERREVT1，ERREVT2，ERREVT3，ERREVT4，ERREVTA はエラーイベントを示します。

SERREVT，SCMPEVT，CERREVT，COPNEVT，CCLSEVT は通信イベントを示します。

注※

MCF アプリケーション定義（mcfaalcap -g）の recvmsg オペランドに r を指定した場合，または dc_mcf_rollback 関数の action に DCMCFRTRY もしくは DCMCFRRTN を指定した場合は除きます。

- **MCF イベント処理用 MHP のアプリケーションの型**

  MCF イベント処理用 MHP のアプリケーションの型は，MCF イベントが通知された原因で決まります。MCF イベント処理用 MHP では，決められたアプリケーションの型に応じた処理をしてください。

  ERREVT1，ERREVT2，および ERREVT3 の MCF イベント処理用 MHP を起動する場合には，**アプリケーション起動プロセス**が必要となります。アプリケーション起動プロセスを使う場合は，MCF 通信構成定義を作成しておいてください。

  dc_mcf_execap 関数で複数の MHP を起動させた場合に MCF イベントが通知されたときは，最初に dc_mcf_execap 関数を呼び出した MHP の型を基に，MCF イベント処理用 MHP の型が決定されます。SPP から dc_mcf_execap 関数を呼び出した場合は，アプリケーション起動プロセスに対応する MCF イベントが通知されます。

  MCF イベント処理用 MHP とアプリケーションの型の関係を次の表に示します。

表 3-14　MCF イベント処理用 MHP とアプリケーションの型の関係

MCF イベント処理用 MHP を起動した MCF イベントのイベントコード	MCF イベント処理用 MHP のアプリケーションの型
ERREVT1	要求元となった論理端末の端末タイプの型に応じて設定されます。 • reply 型論理端末の場合：ans • reply 型以外の論理端末の場合：noans
ERREVT2	MCF イベントが通知される原因となった，MHP のアプリケーションの型をそのまま引き継ぎます。※
ERREVT3	
ERREVT4	
ERREVTA	非応答型（noans）が設定されます。
SERREVT	
SCMPEVT	
CERREVT	
VERREVT	
COPNEVT	

MCF イベント処理用 MHP を起動した MCF イベントのイベントコード	MCF イベント処理用 MHP のアプリケーションの型
VOPNEVT	非応答型（noans）が設定されます。
CCLSEVT	
VCLSEVT	

注※

非トランザクション属性の MHP の場合は，異常終了したあとでもアプリケーションの型を引き継がないで，MCF イベント処理用 MHP の指定に従います。

- 通信プロトコル対応製品と，通知される MCF イベントの関係

通信プロトコル対応製品と，通知される MCF イベントの関係を以降の表に示します。

表 3–15　通信プロトコル対応製品と通知される MCF イベントの関係 1

MCF イベント	通信プロトコル対応製品			
	TP1/NET/OSAS−NIF	TP1/NET/OSI-TP	TP1/NET/SLU-TypeP2	TP1/NET/TCP/IP
ERREVT1	○	○	○	○
ERREVT2	○	○	○	○
ERREVT3	○	○	○	○
ERREVT4	×	○	○	○
ERREVTA	○	○	○	○
SERREVT	×	×	×	×
SCMPEVT	×	×	×	○
CERREVT	○	○	○	○
COPNEVT	○	○	○	○
CCLSEVT	○	○	○	○
VERREVT	×	×	×	×
VOPNEVT	×	×	×	×
VCLSEVT	×	×	×	×

（凡例）

○：該当する通信プロトコル対応製品で通知されます。

×：該当する通信プロトコル対応製品では通知されません。

表 3-16　通信プロトコル対応製品と通知される MCF イベントの関係 2

MCF イベント	通信プロトコル対応製品		
	TP1/NET/User Agent	TP1/NET/UDP	TP1/NET/XMAP3
ERREVT1	○	○	○
ERREVT2	○	○	○
ERREVT3	○	○	○
ERREVT4	○	○	○
ERREVTA	○	○	○
SERREVT	×	×	○※
SCMPEVT	×	×	○※
CERREVT	○	○	×
COPNEVT	○	○	×
CCLSEVT	○	○	×
VERREVT	×	×	○
VOPNEVT	×	×	○
VCLSEVT	×	×	○

（凡例）

○：該当する通信プロトコル対応製品で通知されます。

×：該当する通信プロトコル対応製品では通知されません。

注※

SERREVT，SCMPEVT が通知されるのは，印刷機能を使用する場合だけです。

3.10.1　不正アプリケーション名検出通知イベント（ERREVT1）

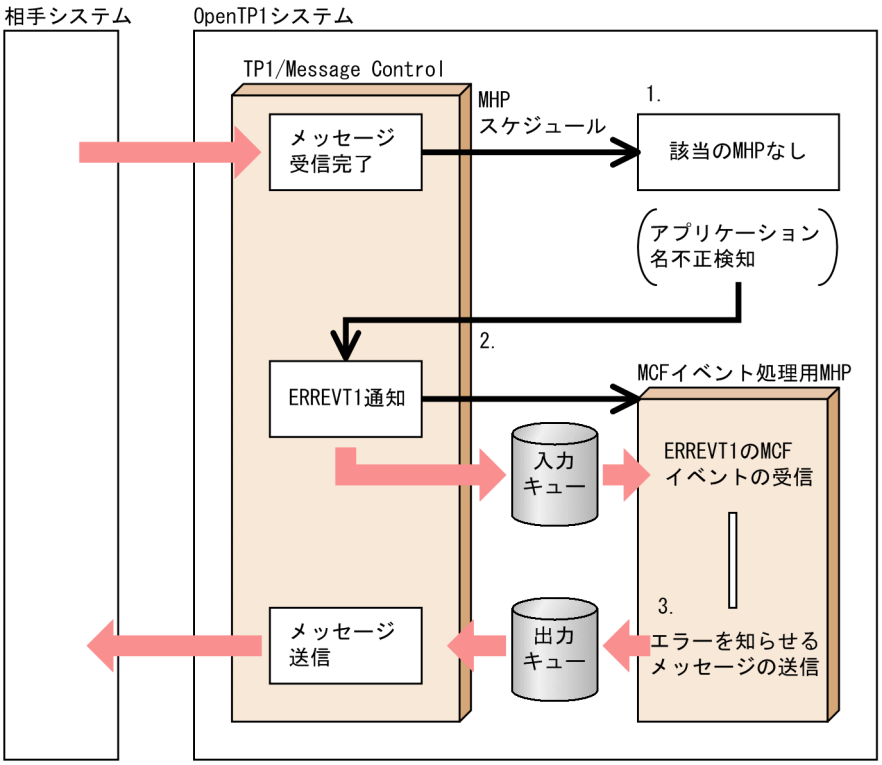
ERREVT1 は，受信したメッセージに設定されたアプリケーション名に，次に示す不良がある場合に通知されます。

- アプリケーション名の形式が間違っている
- アプリケーション名が，MCF アプリケーション定義にない

ERREVT1 の MCF イベント処理用 MHP では，アプリケーション名が自ノードになかったことを伝える一方送信メッセージを送信するなどの対処をしてください。その際は，論理端末や UAP の型に従って，応答メッセージ，または一方送信メッセージを MCF イベント処理用 MHP から送信してください。

ERREVT1 の概要を次の図に示します。

図 3-24　ERREVT1 の概要



1. メッセージを受信した MCF が，アプリケーション名に該当する MHP をスケジュールしようとしましたが，該当する MHP がありませんでした。
2. 制御が MCF に戻り，ERREVT1 が通知されて，ERREVT1 の MCF イベント処理用 MHP がスケジュールされます。
3. MCF イベント処理用 MHP から，メッセージに該当する MHP がないことを伝える一方送信メッセージを送信します。

3.10.2　メッセージ廃棄通知イベント（ERREVT2）

ERREVT2 は，次に示すことが原因で，受信したメッセージを廃棄した場合に通知されます。また，アプリケーション属性定義 mcfaalcap の-n オプションに errevt=yes（通信イベント障害時にエラーイベント通知する）を指定している通信イベントが，次に示す原因で障害が発生した場合にも，ERREVT2 が通知されます。

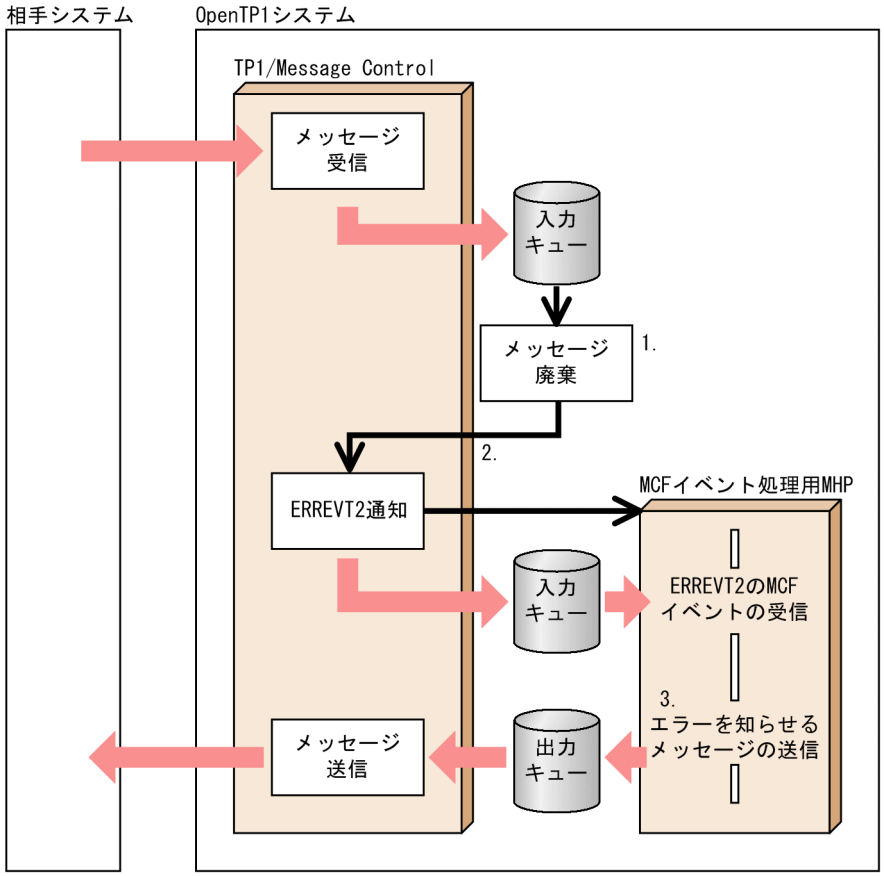
- 入力キューに障害が発生した
- 入力メッセージ最大格納数を超過した
- 動的共用メモリが不足した
- キューファイルが満杯になった
- サービス，サービスグループ，またはアプリケーションが閉塞している

- スケジュール閉塞されているサービスグループの入力キューに未処理受信メッセージが残った状態で，OpenTP1 を正常終了または計画停止 A で終了した
- サービスグループ，またはアプリケーションがセキュア状態である
- MHP で呼び出す dc_mcf_receive 関数にセグメントを渡す前に，MHP が異常終了した
- アプリケーション名に相当する，MHP のサービスがない
- ユーザサーバ未起動などによって，MHP の起動に失敗した
- DBMS の障害などによって，トランザクションの開始に失敗した

ERREVT2 の MCF イベント処理用 MHP では，ERREVT2 の内容を参照して，自ノードで処理できなかったことを伝えるメッセージを送信するなどの対処をしてください。その際は，論理端末や UAP の型に従って，応答メッセージ，または一方送信メッセージを MCF イベント処理用 MHP から送信してください。

ERREVT2 の概要を次の図に示します。

図 3-25　ERREVT2 の概要



1. 受信したメッセージが，何らかの理由で入力キューから廃棄されました。
2. 制御が MCF に戻り，ERREVT2 が通知されて，ERREVT2 の MCF イベント処理用 MHP がスケジュールされます。
3. MCF イベント処理用 MHP からメッセージを送ってきた他システムに，メッセージの再送要求などを伝える一方送信メッセージを送信します。

3.10.3　UAP 異常終了通知イベント（ERREVT3）

ERREVT3 は，次に示す場合に通知されます。また，アプリケーション属性定義 mcfaalcap の-n オプションに errevt=yes（通信イベント障害時にエラーイベント通知する）を指定している通信イベントが，次に示す原因で障害が発生した場合にも，ERREVT3 が通知されます。

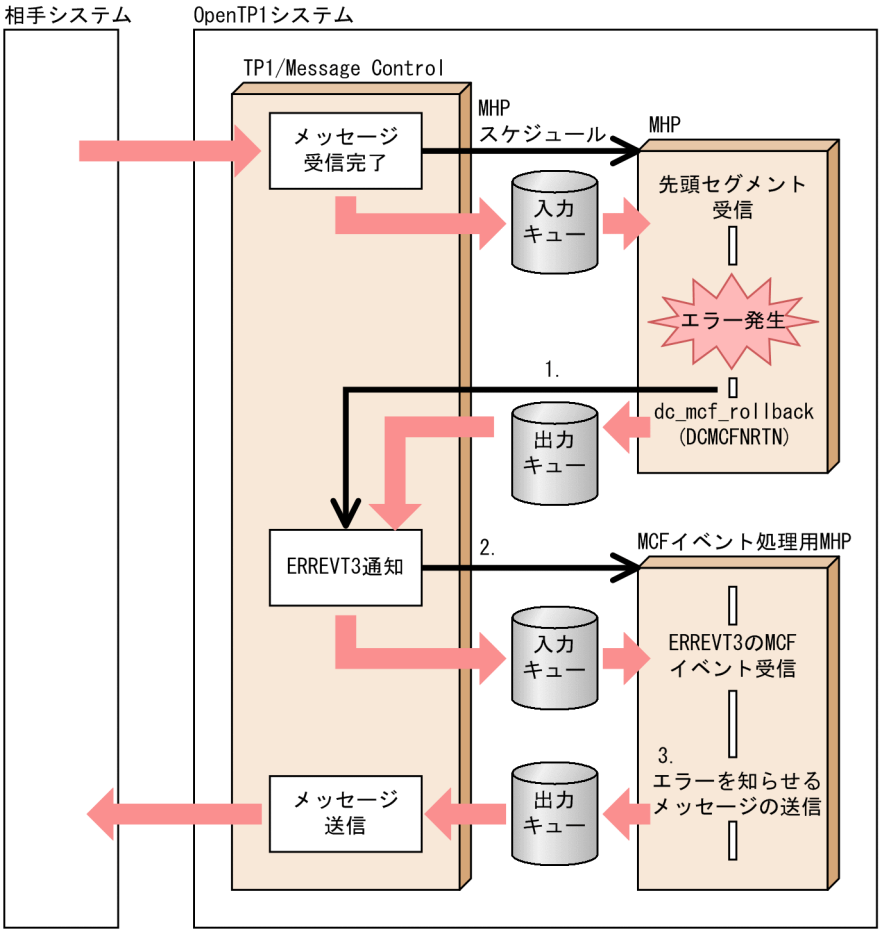
- MHP が，dc_mcf_receive 関数で先頭セグメントを受信したあとで異常終了した
- アプリケーション終了時，障害が発生した

ERREVT3 は，MHP で呼び出した dc_mcf_rollback 関数のフラグに DCMCFNRTN を設定してある場合にも通知されます。

ERREVT3 の MCF イベント処理用 MHP では，ERREVT3 の内容を参照して，自ノードの UAP が異常終了したことを伝えるメッセージのアプリケーション名をキーとして送信するなどの対処をしてください。その際は，論理端末や UAP の型に従って，応答メッセージ，または一方送信メッセージを MCF イベント処理用 MHP から送信してください。

ERREVT3 の概要を次の図に示します。

図 3-26　ERREVT3 の概要



1. メッセージを受信した MHP の処理でエラーが起こると，ロールバックでリトライを設定していない場合，出力キューを経由して制御が MCF に戻ります。
2. エラーが発生した MHP から送られて来た情報を基に，ERREVT3 が通知されます。
3. ERREVT3 は入力キューを経由して，ERREVT3 の MCF イベント処理用 MHP をスケジュールします。MCF イベント処理用 MHP は，メッセージを送ってきた他システムに，自システムの UAP でエラーが起こったことを伝えるメッセージを送信します。

3.10.4　タイマ起動メッセージ廃棄通知イベント（ERREVT4）

ERREVT4 は，次に示すことが原因でメッセージを廃棄した場合に通知されます。

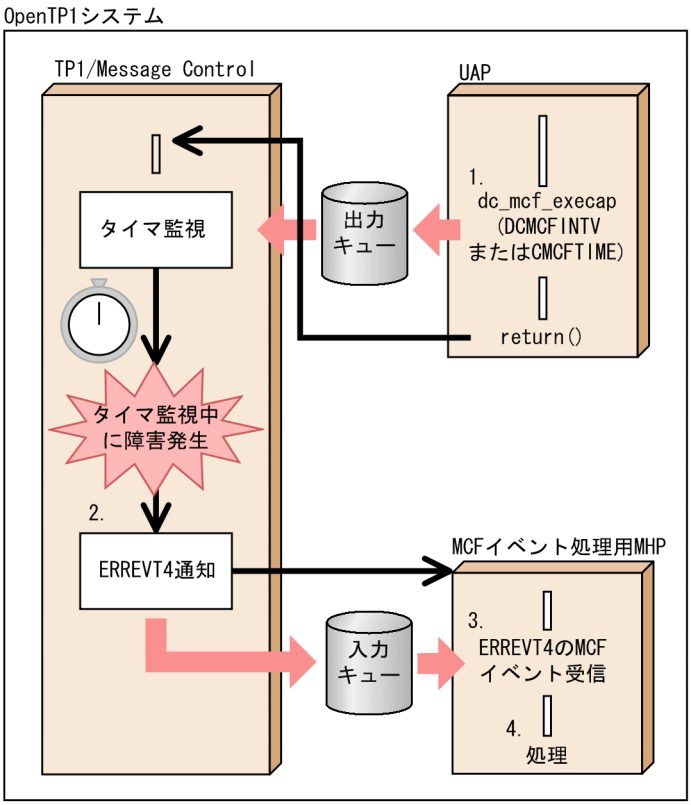
- UAP からタイマ起動のアプリケーション起動（dc_mcf_execap 関数）をして，MCF でタイマ監視中に，障害でメッセージを廃棄したとき

(1)　ERREVT4 が通知されるまでの流れ

MHP からタイマ起動の dc_mcf_execap 関数を呼び出すと，MCF では出力キューからメッセージを取り出してタイマ監視をします。その時間待ちから入力キューに書き込むまでの間で，タイマ監視の失敗やスケジュールの失敗などの障害が起こると，ERREVT4 が通知されます。その後，ERREVT4 を処理する MCF イベント処理用 MHP が起動されます。

ERREVT4 の概要を次の図に示します。

図 3-27　ERREVT4 の概要



1. タイマ起動の dc_mcf_execap 関数を呼び出して，そのトランザクションがコミットすると，MCF でタイマ監視を始めます。
2. MCF でタイマ監視中に障害が起こると，ERREVT4 が通知されます。
3. ERREVT4 は入力キューを経由して，ERREVT4 の MCF イベント処理用 MHP をスケジュールします。
4. ERREVT4 の MCF イベント処理用 MHP では，ERREVT4 を解析して対処します。

3.10.5　未処理送信メッセージ廃棄通知イベント（ERREVTA）

ERREVTA は，次に示す場合に通知されます。

- OpenTP1 の正常終了コマンド（dcstop コマンド）を実行したあとに，出力キューに残っているメッセージを廃棄したとき
- OpenTP1 が稼働している間に，未処理送信メッセージが残っている出力キューを mcftdlqle コマンドまたは dc_mcf_tdlqle 関数で削除したとき
- タイマ起動の dc_mcf_execap 関数を呼び出して，タイマ監視中に OpenTP1 の正常終了コマンド（dcstop コマンド）を実行したとき

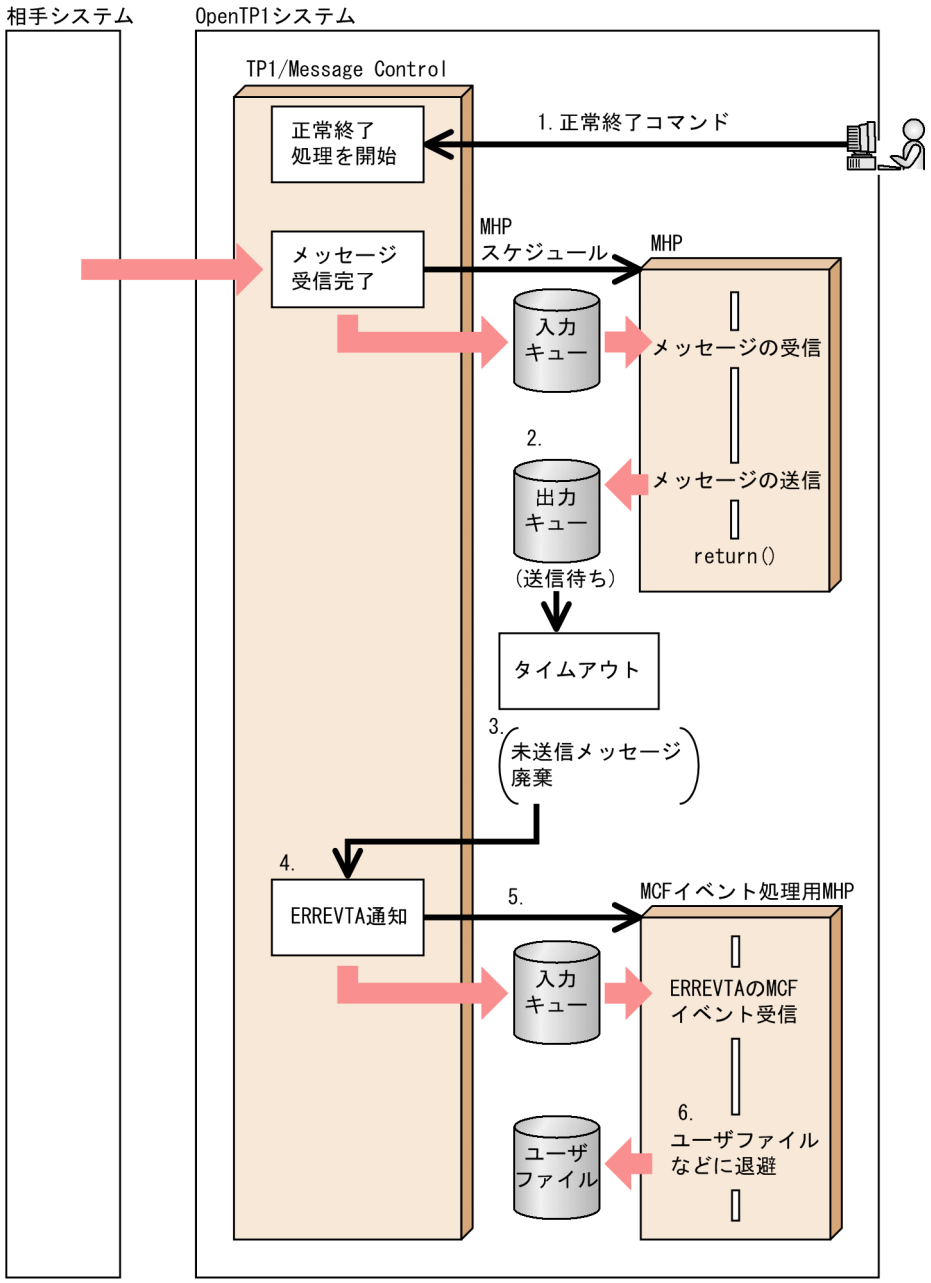
(1) ERREVTA が通知されるまでの流れ

MHP が正常終了すると，出力キューに送信するメッセージが出力されます。閉塞解除されている論理端末において，メッセージが送信待ちしている状態で，OpenTP1 を正常終了する場合，出力キューにある送信メッセージを送信し終わるまで MCF は終了を待ちます。このとき，送信する先のシステムの障害などで送信できない場合，時間切れ（タイムアウト）となり，送信するメッセージを廃棄します。このメッセージの廃棄を知らせるために ERREVTA が通知されます。タイムアウトになる時間は，タイマ定義（mcfttim -t）の mtim オペランドに指定した値で監視しています。

ただし，dc_mcf_execap 関数によるタイマ起動要求メッセージや閉塞されている論理端末の出力キューに残っている未処理送信メッセージは，未処理送信メッセージ滞留時間の監視対象にはなりません。そのため，OpenTP1 の正常終了コマンド（dcstop コマンド）を実行すると，メッセージはすぐに破棄され，ERREVTA が通知されます。

ERREVTA の概要を次の図に示します。

図 3-28　ERREVTA の概要



1. OpenTP1 の正常終了コマンド（dcstop コマンド）を実行します。タイマ起動要求メッセージや閉塞されている論理端末の出力キューに未処理送信メッセージが残っていた場合は，このタイミングでメッセージが破棄され，MCF から ERREVTA が通知されます。
2. MHP で正常に処理されたメッセージが出力キューに格納されます。
3. 出力キューのメッセージの送信タイムアウトで，出力するメッセージが廃棄されました。
4. MCF から ERREVTA が通知されます。
5. ERREVTA の MCF イベント処理用 MHP がスケジュールされます。
6. ユーザファイルなどに退避します。

3.10.6　送信障害通知イベント（SERREVT）

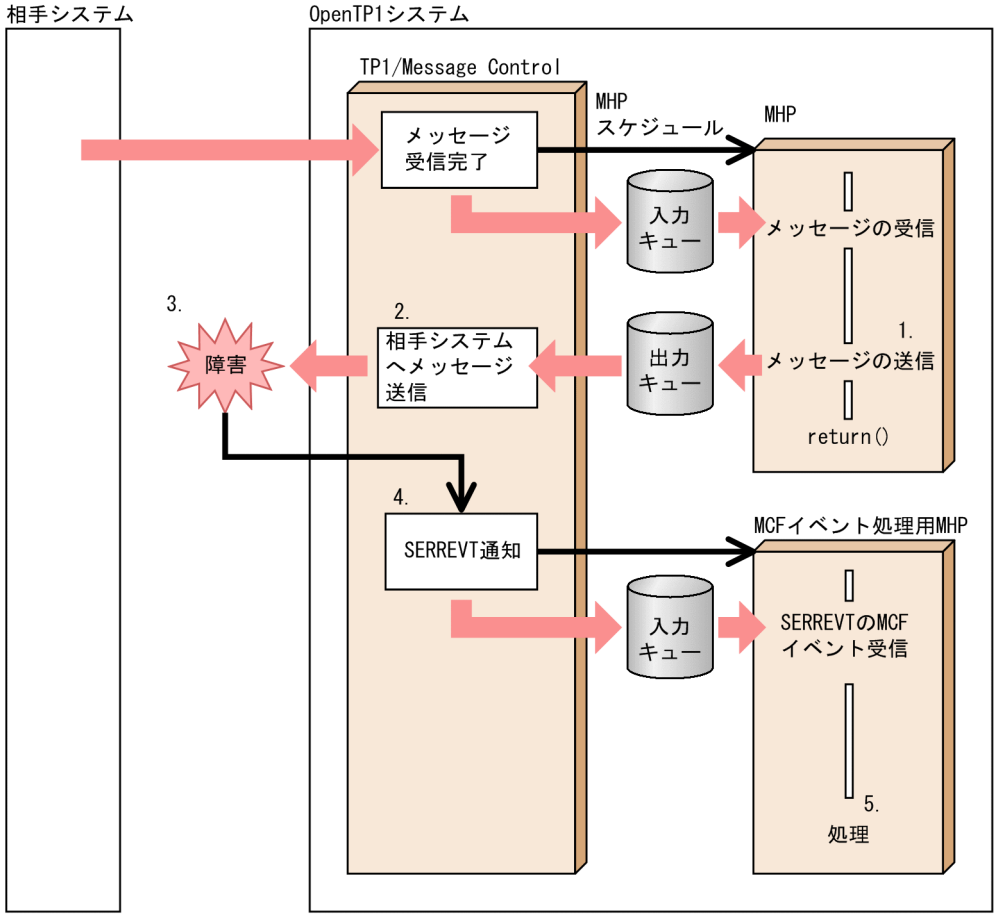
SERREVT は，メッセージを正常に送信した UAP が処理を終了したあとで，MCF が相手システムへメッセージを送信する途中で通信プロトコルの障害が起こった場合に通知されます。このイベントを参照すると，非同期型のメッセージの送信（dc_mcf_send 関数，dc_mcf_reply 関数）を使った場合でも，通信プロトコルの障害が起こったことがわかります。

SERREVT の MCF イベント処理用 MHP は，非応答型（noans 型）です。

イベントを入力キューに書き込む前に OpenTP1 を終了した場合は，SERREVT は通知されません。

SERREVT の概要を次の図に示します。

図 3-29　SERREVT の概要



1. dc_mcf_send 関数，または dc_mcf_reply 関数に「イベントを通知する」ことを引数に設定して，メッセージを送信します。
2. UAP は正常終了します。UAP からの送信要求を受け取った MCF は，メッセージを相手システムへ送信します。
3. 通信プロトコルの障害が起こりました。
4. 制御が MCF に戻り，SERREVT が通知されて，MCF イベント処理用 MHP がスケジュールされます。
5. MCF イベント処理用 MHP では，SERREVT で通知された内容に合わせた処理をします。

3.10.7　送信完了通知イベント（SCMPEVT）

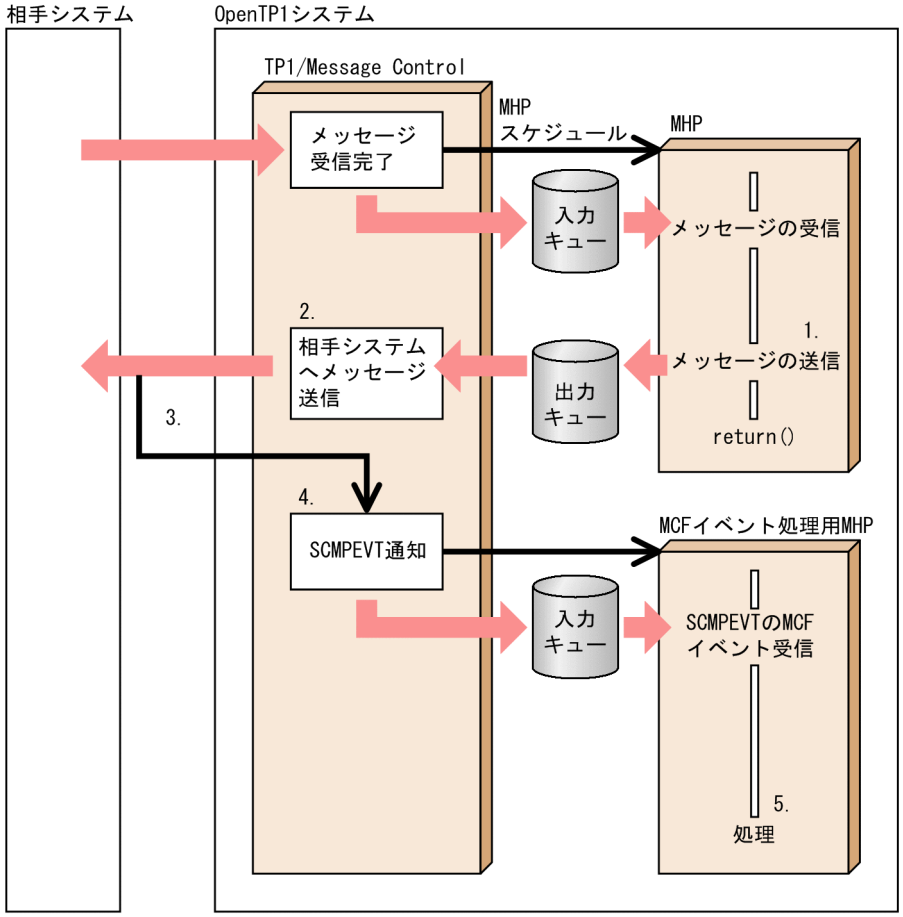
SCMPEVT は，メッセージを正常に送信できた場合に MCF から通知されます。このイベントによって，非同期型のメッセージの送信（dc_mcf_send 関数，dc_mcf_reply 関数）が正常に相手システムまで届いたことがわかります。

SCMPEVT の MCF イベント処理用 MHP では，送信完了と同期させる処理を開始できます。このときの MCF イベント処理用 MHP は，非応答型（noans 型）です。

イベントを入力キューに書き込む前に OpenTP1 を終了した場合は，SCMPEVT は通知されません。

SCMPEVT の概要を次の図に示します。

図 3-30　SCMPEVT の概要



1. dc_mcf_send 関数，または dc_mcf_reply 関数に「イベントを通知する」ことを引数に設定して，メッセージを送信します。
2. UAP からの送信要求を受け取った MCF は，メッセージを相手システムへ送信します。
3. メッセージが相手システムへ正常に送信されました。
4. 制御が MCF に戻り，SCMPEVT が通知されて，MCF イベント処理用 MHP がスケジュールされます。
5. MCF イベント処理用 MHP では，SCMPEVT で通知された内容に合わせた処理をします。

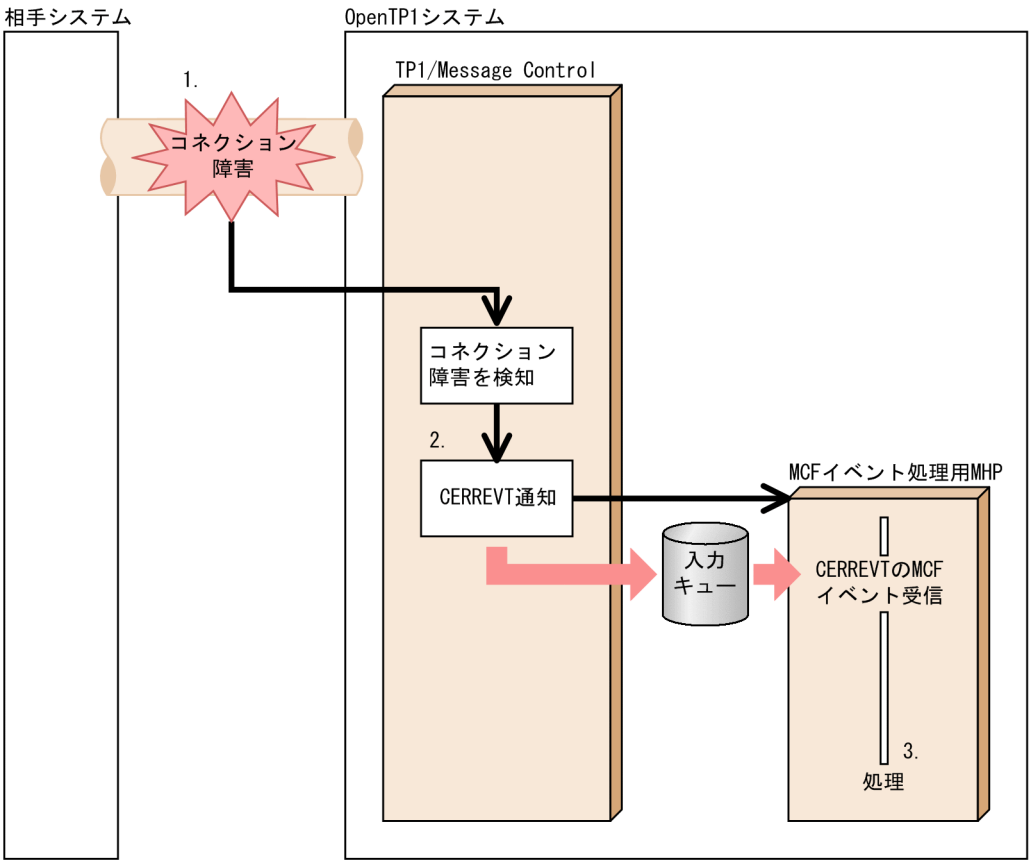
3.10.8　障害通知イベント（CERREVT，VERREVT）

CERREVT（VERREVT）は，通信管理プログラムでコネクション障害，またはLE障害が起こったときに通知されます。ただし，MCF通信構成定義プロトコル固有定義にコネクション確立再試行を指定している場合は，CERREVT（VERREVT）は通知されません。

コネクション障害の通知方法は，プロトコル製品によって異なります。CERREVT（VERREVT）が通知される形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

CERREVT（VERREVT）の概要を次の図に示します。

図 3-31　CERREVT（VERREVT）の概要



1. 相手システムとの通信中に，コネクションに障害が起こりました。
2. コネクション確立再試行を指定していない場合，および再試行回数の指定値を超えた場合，CERREVT（VERREVT）が通知されて，MCFイベント処理用MHPがスケジュールされます。
3. MCFイベント処理用MHPでは，CERREVT（VERREVT）で通知された内容に合わせた処理をします。

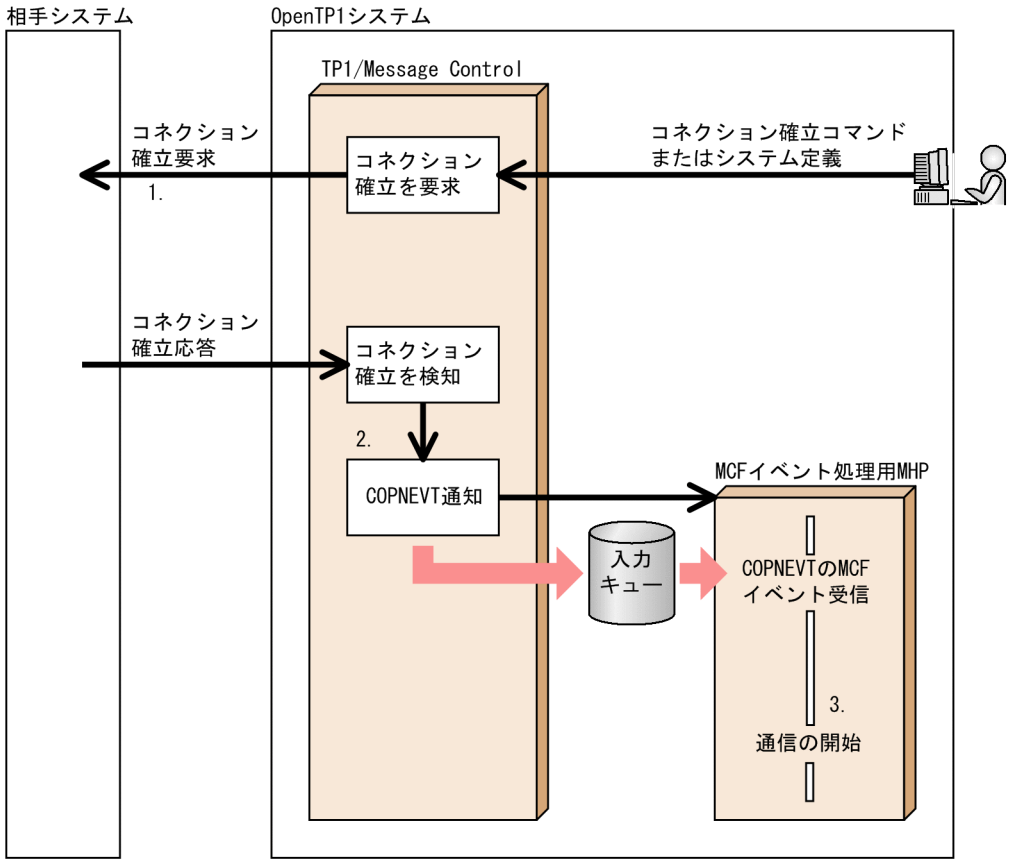
3.10.9　コネクション確立通知イベント（COPNEVT，VOPNEVT）

COPNEVT（VOPNEVT）は，MCF および通信管理プログラムが相手システムとのコネクションを確立したときに通知されます。COPNEVT（VOPNEVT）が通知されることで，コネクションが確立したことが MHP でわかります。

コネクション確立の通知方法は，プロトコル製品によって異なります。COPNEVT（VOPNEVT）が通知される形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

COPNEVT（VOPNEVT）の概要を次の図に示します。

図 3-32　COPNEVT（VOPNEVT）の概要



1. 自 OpenTP1 システム，または相手システムからコネクションの確立を要求して，コネクションを確立します。この例では，自 OpenTP1 システムからコネクションの確立を要求します。プロトコル製品によっては，相手システムから応答が返らない場合もあります。
2. コネクションが確立されると，COPNEVT（VOPNEVT）が通知されて，MCF イベント処理用 MHP がスケジュールされます。
3. MCF イベント処理用 MHP では，COPNEVT（VOPNEVT）で通知された内容に合わせた処理をします。

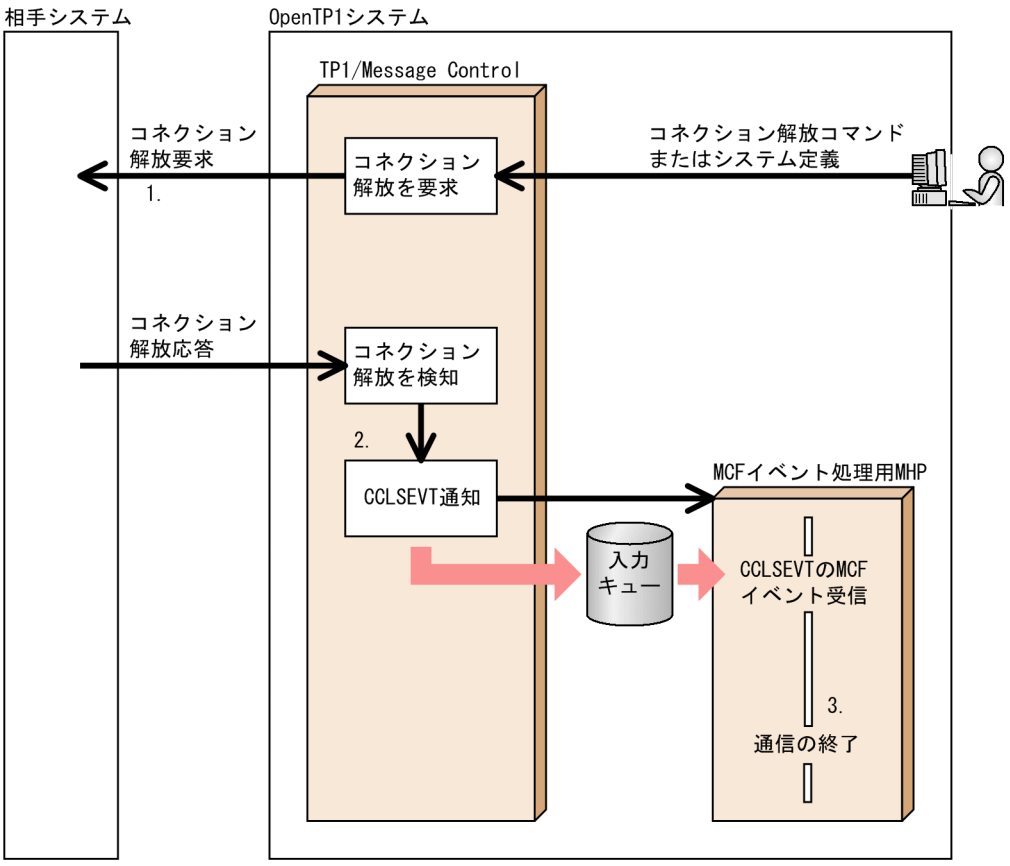
3.10.10　コネクション解放通知イベント（CCLSEVT，VCLSEVT）

CCLSEVT（VCLSEVT）は，MCF および通信管理プログラムが相手システムとのコネクションを解放したときに通知されます。CCLSEVT（VCLSEVT）が通知されることで，コネクションが解放されたことが MHP でわかります。

コネクション解放の通知方法は，プロトコル製品によって異なります。CCLSEVT（VCLSEVT）が通知される形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

CCLSEVT（VCLSEVT）の概要を次の図に示します。

図 3-33　CCLSEVT（VCLSEVT）の概要



1. 自 OpenTP1 システム，または相手システムからコネクションの解放を要求して，コネクションを解放します。この例では，自 OpenTP1 システムからコネクションの解放を要求します。プロトコル製品によっては，相手システムから応答が返らない場合もあります。
2. コネクションが解放されると，CCLSEVT（VCLSEVT）が通知されて，MCF イベント処理用 MHP がスケジュールされます。
3. MCF イベント処理用 MHP では，CCLSEVT（VCLSEVT）で通知された内容に合わせた処理をします。

3.10.11　MCF イベントのメッセージ形式

MCF イベントとして渡される論理メッセージは，**MCF イベント情報**と，**処理できなかったメッセージ**から構成されます。処理できなかったメッセージは，通知された MCF イベントによって異なります。

ERREVT1 の場合
　アプリケーションを特定できなかったメッセージ

ERREVT2 の場合
　アプリケーションの閉塞などで，目的のアプリケーションに渡せなかったメッセージ

ERREVT3 の場合
　異常終了した MHP（アプリケーション）で受信したメッセージ

ERREVT4 の場合
　タイマ起動のアプリケーションプログラムの起動で，開始する MHP に渡そうとしたメッセージ

ERREVTA の場合
　出力キューで滞留していたメッセージ

次に示す MCF イベントの場合は，MCF イベント情報だけが渡されます。処理できなかったメッセージに該当するものはありません。

- SERREVT
- SCMPEVT
- CERREVT（VERREVT）
- COPNEVT（VOPNEVT）
- CCLSEVT（VCLSEVT）

(1)　MCF イベントのメッセージ構造

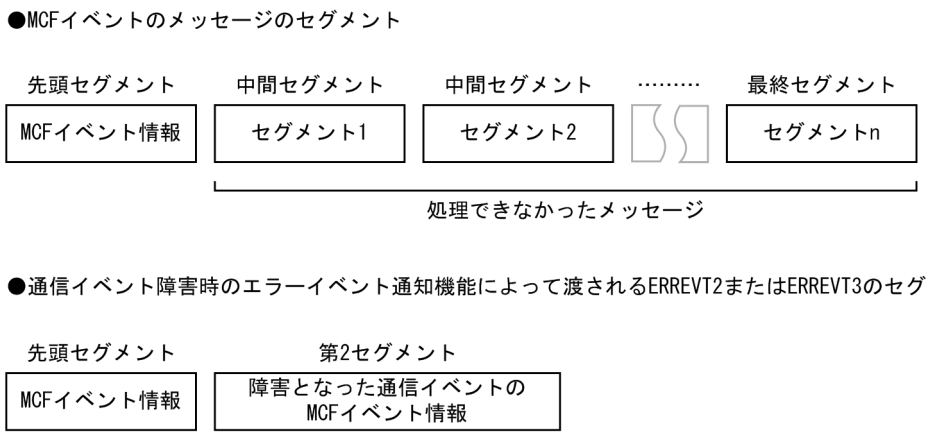
MCF イベントを MCF イベント処理用 MHP で受信する場合は，通常のメッセージを受信する関数（dc_mcf_receive 関数）を使います。

MCF イベントは，複数セグメントの論理メッセージとして，MCF イベント処理用 MHP に渡されます。先頭セグメントには MCF イベント情報が，2 番目以降のセグメントには処理できなかったメッセージのセグメントが設定されています。このとき，元の先頭セグメントは 2 番目のセグメントになり，以降一つずつずれて MCF イベントのセグメントとなります。

また，通信イベント障害時のエラーイベント通知機能（アプリケーション属性定義 mcfaalcap の-n オプションに errevt=yes を指定）によって起動された MCF イベント処理用 MHP に ERREVT2 または ERREVT3 が渡されます。先頭セグメントには MCF イベント情報が，2 番目のセグメントに障害となった通信イベントの MCF イベント情報が設定されています。

MCF イベントとして渡される論理メッセージのセグメント形式を次の図に示します。

図 3-34　MCF イベントとして渡される論理メッセージのセグメント



(2)　通知されるデータ形式

MCF イベント情報は，MCF イベント処理用 MHP をコーディングした高級言語（C 言語，または COBOL 言語）に合わせて通知されます。

C 言語の MHP では，MCF イベント情報を**構造体**で受け取れます。この構造体はヘッダファイル <dcmcf.h>で定義されています。MCF イベント情報を処理する MHP では，＃include で<dcmcf.h>をインクルードしてください。また，通信イベントによっては，通信プロトコル製品別のヘッダファイルに，構造体を定義している場合もあります。

COBOL 言語の MHP では，MCF イベント情報を**セグメントの並び**で受け取れます。このセグメントのバイト位置で必要な情報を取り出します。

MCF イベント情報のデータ形式は，通信プロトコル対応製品（TP1/NET/××××）によって異なります。次に示す MCF イベント情報のデータ形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

- ERREVT1
- ERREVT2
- ERREVT3
- ERREVTA
- SERREVT
- SCMPEVT
- CERREVT（VERREVT）
- COPNEVT（VOPNEVT）
- CCLSEVT（VCLSEVT）

ERREVT4 の MCF イベント情報のデータ形式については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。

3.11　アプリケーションプログラムが使う MCF のプロセス

UAP が使う MCF のプロセスについて説明します。UAP がメッセージで通信するときには，次に示す MCF サービスのプロセスを使います。

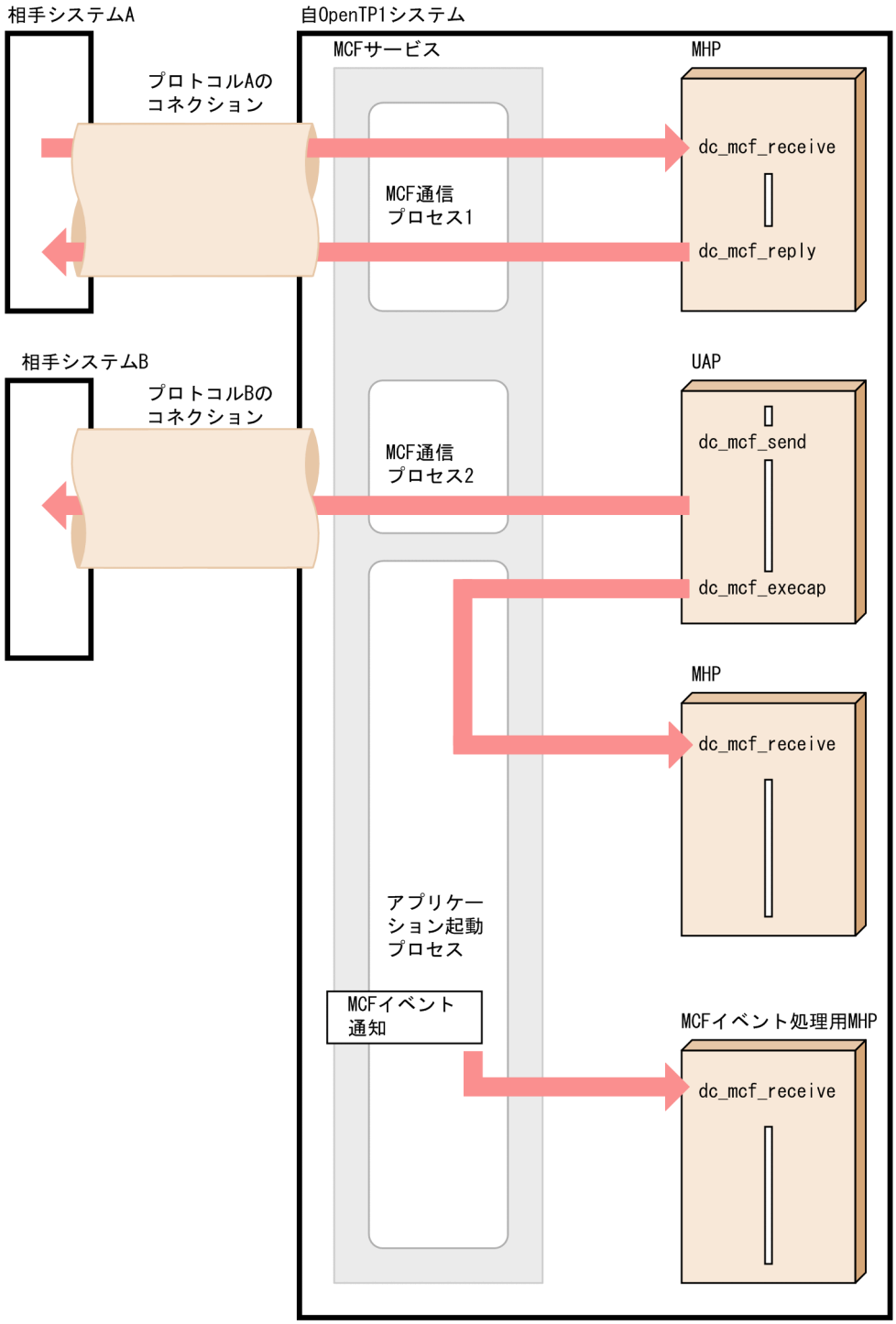
- **MCF 通信プロセス**
  自 OpenTP1 システムが，相手システムと通信するときに使うシステムプロセスです。
- **アプリケーション起動プロセス**
  OpenTP1 システム内部でメッセージをやり取りするときに使うシステムプロセスです。

MCF のプロセスを上記のように分ける理由を，次に示します。

1. 外部との通信と内部での通信でシステムプロセスを分けることで，MCF の通信に使うプロセスの負担を分散できます。
2. 通信プロトコルの障害が原因で MCF 通信プロセスが使えない場合でも，内部処理のプロセスに影響するのを防げます。

UAP で使う MCF のプロセスの概要を次の図に示します。

図 3-35　UAP で使う MCF のプロセスの概要



3.11.1　MCF のプロセスの種類

MCF のプロセス（MCF 通信プロセス，アプリケーション起動プロセス）について説明します。

(1) MCF 通信プロセス

MCF 通信プロセスは，**自 OpenTP1 システムと相手システムが通信するとき**に使うプロセスです。UAP が次に示す関数で相手システムと通信するときに，MCF 通信プロセスを使います。

- メッセージの受信（dc_mcf_receive 関数）
- メッセージの送信（dc_mcf_send 関数）
- 応答メッセージの送信（dc_mcf_reply 関数）
- 同期型のメッセージの受信（dc_mcf_recvsync 関数）
- 同期型のメッセージの送信（dc_mcf_sendsync 関数）
- 同期型のメッセージの送受信（dc_mcf_sendrecv 関数）
- メッセージの再送（dc_mcf_resend 関数）

MCF 通信プロセスは，**通信プロトコル対応製品ごと**に作成します。一つの OpenTP1 システムで異なる複数の通信プロトコルを使って通信する場合は，それぞれの通信プロトコル対応製品ごとに，MCF 通信プロセスを定義します。

(2) アプリケーション起動プロセス

アプリケーション起動プロセスは，**自 OpenTP1 システム内部の MHP にメッセージを渡すとき**に使います。

アプリケーション起動プロセスを使うのは，次に示す機能を実行した場合です。

- アプリケーションプログラムの起動（dc_mcf_execap 関数）
- MHP のロールバック（dc_mcf_rollback 関数）でリトライ指定をした場合
- 通知される MCF イベントのうち，エラーイベント（ERREVT×）を受信して業務で使う場合
- MHP を mcfuevt コマンドで開始する場合
- 異常終了した MHP を再スケジュールする場合（アプリケーション属性定義（mcfaalcap）または UAP 共通定義（mcfmuap）の reschedulecnt オペランドの指定値が 1 以上）

アプリケーション起動プロセスは，ほかのシステムとの通信では使いません（通信プロトコルに依存しません）。通常，アプリケーション起動プロセスはノードに一つ定義します。

3.11.2 MCF のプロセスを使うためのファイル

MCF サービスを使うためには，次に示すファイルを準備する必要があります。

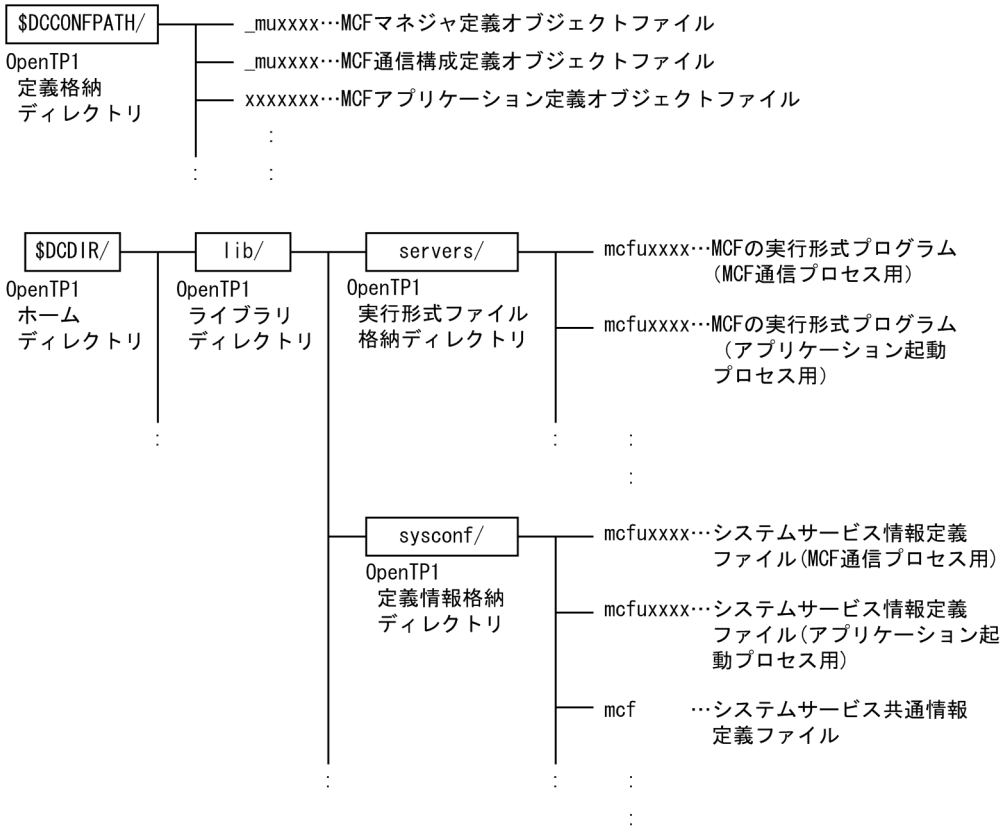
- 定義オブジェクトファイル
- MCF の実行形式プログラム
- システムサービス情報定義ファイル

MCF 通信プロセスの場合，通信プロトコル対応製品によって，定義する内容や生成するコマンドの文法が異なります。MCF 通信プロセスに関する定義方法や定義コマンドユティリティについては，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編の説明を参照してください。

アプリケーション起動プロセスの場合は，通信プロトコルに依存しないで作成できます。アプリケーション起動プロセスに関する定義方法や定義コマンドユティリティについては，マニュアル「OpenTP1 システム定義」および「OpenTP1 運用と操作」を参照してください。

MCF サービスに必要なファイルのディレクトリ構成を，次の図に示します。

図 3-36　MCF サービスに必要なファイルのディレクトリ構成



4 ユーザデータを使う場合の機能

OpenTP1 のファイルサービス（TP1/FS/Direct Access，TP1/FS/Table Access）を使う場合の機能，IST サービス（TP1/Shared Table Access），ISAM ファイルを使う場合の機能，他社のデータベースを使う場合の注意，および任意のファイルなどの資源を排他制御する場合の機能について説明します。

この章では，各機能を **C 言語の関数名**で説明します。C 言語の関数名に該当する COBOL 言語の API は，関数を最初に説明する個所に【 】で囲んで表記します。それ以降は，C 言語の関数名に統一して説明します。

COBOL 言語の API がない関数の場合は，【 】の表記はしません。

4.1 DAM ファイルサービス（TP1/FS/Direct Access）

OpenTP1 専用のユーザファイルとして使える，**DAM ファイル**について説明します。DAM ファイルを使う場合は，システムに TP1/FS/Direct Access が組み込まれていることが前提となります。また，OpenTP1 の基本機能が TP1/Server Base の場合だけ使えます。TP1/LiNK では，DAM ファイルサービスは使えません。

4.1.1 DAM ファイルの構成

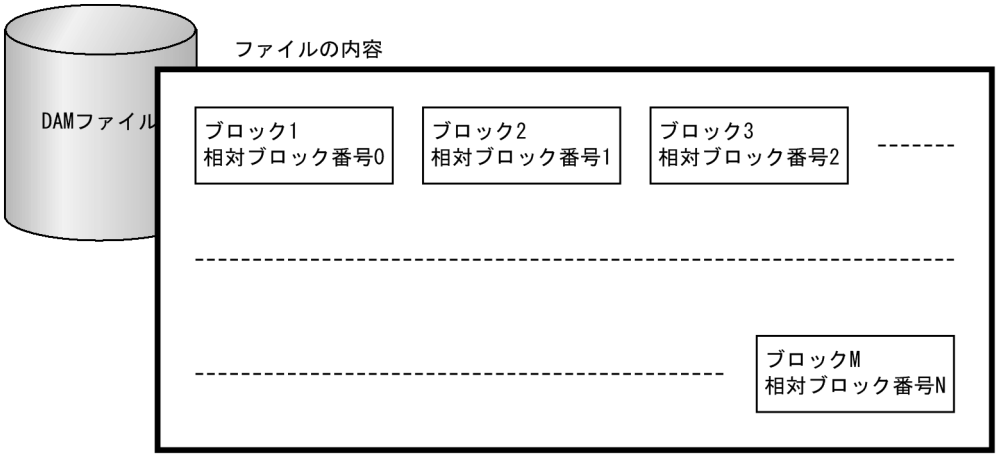
DAM ファイルを構成する一つ一つのデータを**ブロック**といいます。DAM ファイルの 1 ブロックの大きさは，セクタ長×n−8（n は正の整数）バイトで指定します。

セクタ長は次のとおりです。

- キャラクタ型スペシャルファイルの場合：filmkfs コマンドの-s オプション指定値
- 通常ファイルの場合：512 バイト

DAM ファイルの構成を次の図に示します。

図 4–1 DAM ファイルの構成



4.1.2 物理ファイルと論理ファイル

UAP から呼び出す関数に指定する，DAM ファイルの名称について説明します。

(1) 物理ファイル

DAM ファイルを作成する場合は，DAM ファイルとして使う領域を damload コマンドまたはオフラインの業務をする UAP で割り当てます。割り当てるときに使用する名称は，OpenTP1 ファイルシステムとして割り当てたキャラクタ型スペシャルファイルのパス名です。このパス名で示すファイルを**物理ファイ**

ル，ファイル名を**物理ファイル名**といいます。物理ファイル名は，割り当てた DAM ファイルに初期データを作成したり，オフライン環境で更新したりする場合に使います。

(2) 論理ファイル

オンライン中に SUP，SPP，および MHP から DAM ファイルを使うときは，システム定義で名称を付けた，論理的なファイルに対してアクセスします。このファイルを**論理ファイル**，ファイル名を**論理ファイル名**といいます。論理ファイル名と物理ファイル名は，1 対 1 に対応しています。物理ファイル名と論理ファイル名の対応は，DAM サービス定義で指定します。

(3) UAP からアクセスする場合の注意

論理ファイルと物理ファイルはノードごとに定義するので，DAM ファイルはノードごとに独立して管理されています。そのため，UAP からほかのノードにある DAM ファイルへはアクセスできません。DAM ファイルを使う場合は，ノード（マシン）にある UAP の処理から，ノード内で定義したファイル名でアクセスしてください。

4.1.3 DAM ファイルへのアクセスの概要

DAM ファイルにアクセスする方法について説明します。DAM ファイルには，次の 2 種類があります。

- **回復対象の DAM ファイル**
  UAP のトランザクション処理と同期してブロックを入出力します。
- **回復対象外の DAM ファイル**
  トランザクション処理とは関係なくブロックを入出力します。

以降，回復対象の DAM ファイル固有の機能については「**回復対象の DAM ファイルの場合**」という断りを入れて説明します。回復対象外の DAM ファイルについては，「4.1.8 回復対象外の DAM ファイルへのアクセス」を参照してください。

(1) DAM ファイルへのアクセス手順の概要

UAP から DAM ファイルを使う場合は，次の手順でアクセスします。

1. ファイルをオープンします。
2. 次に示すうち，どれか一つの処理をします。
   データの入力（参照），データの入力と更新，データの出力
3. ファイルをクローズします。

(2) DAM ファイルをオープンする場合の注意

DAM ファイルをオープンする関数は，DAM ファイルを使う UAP ごとに呼び出します。同じグローバルトランザクションに属していても，DAM ファイルのオープンは UAP ごとに呼び出してください。

- 回復対象の DAM ファイルの場合

  論理ファイルのオープンは，トランザクションの範囲内でも範囲外でもできます。ただし，次に示すときは，論理ファイルにはブロック排他だけ指定できます。

  - トランザクションを開始する前に論理ファイルをオープンするとき
  - グローバルトランザクション単位で排他制御する指定をしたとき

(3) DAM ファイルからブロックを入力する場合の注意

DAM ファイルのブロックは，参照目的の入力のときにかぎり，排他を指定しないで入力できます。ただし，排他を指定しないでブロックを入力した場合，入力処理中にほかの UAP から該当するブロックが更新されるため，最新のブロックの内容を入力できないことがあります。

(4) トランザクションの範囲外からの DAM ファイルへのアクセス（回復対象の DAM ファイルの場合）

トランザクションの範囲外（トランザクションを開始する前，および同期点取得後）の処理からは，ブロックの入力だけできます。その場合は，参照目的のアクセスだけで，排他を指定できません。

(5) DAM ファイルへアクセスするトランザクションの範囲（回復対象の DAM ファイルの場合）

DAM ファイルへアクセスするときは，トランザクションブランチ単位で排他制御します。また，グローバルトランザクション単位でも排他制御できます。DAM ファイルの排他については，「4.1.7　DAM ファイルの排他制御」を参照してください。

(6) 2GB の容量を超える DAM ファイルをアクセスする場合の注意

DAM の入出力関数（dc_dam_get，dc_dam_put，dc_dam_read，dc_dam_rewrite，dc_dam_write，CBLDCDMB('GET')，CBLDCDMB('PUT')，CBLDCDAM('READ')，CBLDCDAM('REWT')，CBLDCDAM('WRIT')）で 2GB を超える DAM ファイルデータを一度に入出力できません。2GB を超える DAM ファイルデータを一度に入出力しようとした場合は，DCDAMER_BUFER，または 01604 でエラーリターンします。

4.1.4 オンライン中の DAM ファイルのアクセス（SUP，SPP，MHP からの操作）

オンライン中に，UAP から DAM ファイルにアクセス（ファイルの参照や更新）するときは，トランザクションの中での処理が前提です。トランザクションを開始する前に DAM ファイルをオープンした場合は，オープン以降に開始したすべてのトランザクションを終了させてから DAM ファイルをクローズしてください。

(1) DAM ファイルにアクセスするときの名称

DAM ファイルは，論理ファイル名を指定した dc_dam_open 関数【CBLDCDAM('OPEN')】でオープンします。DAM ファイルをオープンしたときに，そのファイルを識別する名称として**ファイル記述子**がリターンされます。オープン以降の処理（ファイルの入力，更新，出力）では，このファイル記述子を関数に設定してアクセスします。DAM ファイルのクローズでもファイル記述子を設定した dc_dam_close 関数【CBLDCDAM('CLOS')】でクローズします。ファイル記述子は，オープン以降の処理でも UAP 内で保持しておいてください。

ファイル記述子が有効になる範囲を次に示します。

- **トランザクションの範囲内でファイルをオープンした場合**
  ファイル記述子は，次に示すことが起こるまで有効です。
  - 論理ファイルのクローズ
  - トランザクションの同期点取得
  - UAP のプロセスの終了
- **トランザクションの範囲の外でファイルをオープンした場合**
  ファイル記述子は，次に示すことが起こるまで有効です。
  - 論理ファイルのクローズ
  - UAP のプロセスの終了

処理の途中で DAM ファイルを論理閉塞，閉塞を解除，およびファイルの状態を参照するときは，論理ファイル名を使います。

(2) 複数ブロックの一括入出力

複数の連続するブロックを，一括して入出力できます。DAM ファイルを入出力するときに，アクセスするブロックの範囲を構造体として関数に設定します。ブロックの範囲は，**相対ブロック番号**で設定します。この構造体は複数個指定できます。

(3) ブロックの参照／更新手順

DAM ファイルのブロックを参照するときは，dc_dam_read 関数【CBLDCDAM('READ')】でブロックを入力します。そのとき，ほかのトランザクションからの参照または更新を許すかどうかを設定できます。

DAM ファイルのブロックの更新は，dc_dam_read 関数でブロックを入力したあとで，そのブロックを更新するために dc_dam_rewrite 関数【CBLDCDAM('REWT')】を呼び出します。

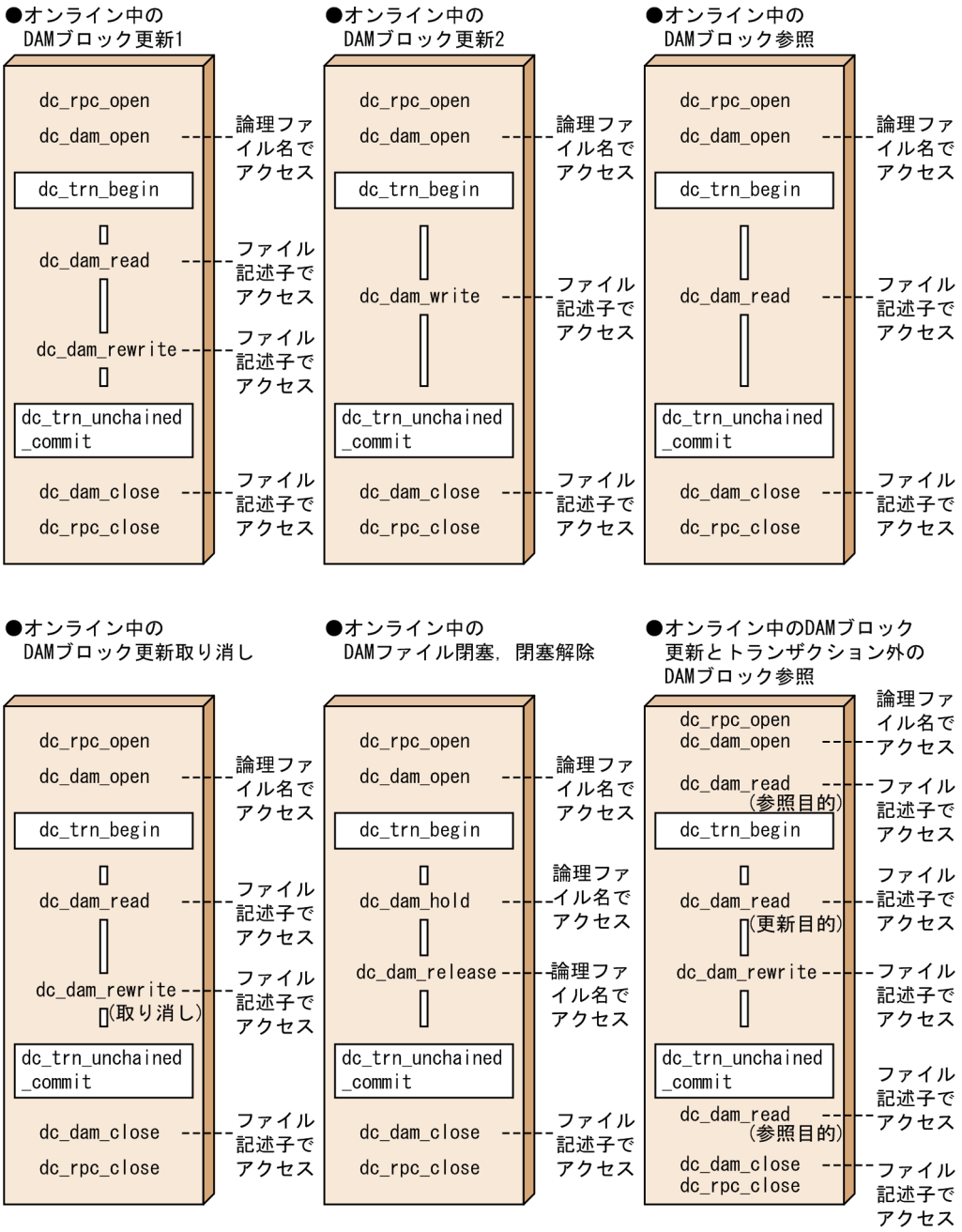
DAM ファイルからブロックを入力しないで，ブロックに上書きする出力をする場合は，dc_dam_write 関数【CBLDCDAM('WRIT')】を呼び出します。

(4) DAM ファイルの論理閉塞と解除

DAM ファイルのブロックを処理中に論理的な矛盾を発見した場合，その DAM ファイルに対してほかの UAP がアクセスできないように，UAP から dc_dam_hold 関数【CBLDCDAM('HOLD')】を呼び出して閉塞できます。また，UAP から dc_dam_release 関数【CBLDCDAM('RLES')】を呼び出して閉塞を解除できます。

オンライン中の DAM ファイルのアクセス手順を次の図に示します。

図 4-2　オンライン時の DAM ファイルのアクセス



(5)　トランザクション処理からの DAM ファイルブロックへのアクセス（回復対象の DAM ファイルの場合）

トランザクションの処理から DAM ファイルにアクセスしてエラーが起こったときは，UAP から abort() を呼び出してトランザクションのプロセスを異常終了させてください。

以前にアクセスした関数によっては，同じトランザクション内からでも，一つのブロックに対するアクセスがエラーリターンすることがあります。エラーリターンするかどうかは，一つのトランザクション内から関数を呼び出したときと，異なるトランザクションから関数を呼び出したときで異なります。以前に呼び出した関数と DAM ファイルのブロックにアクセスできる関数を表 4-1 と表 4-2 に示します。

表 4-1　一つのトランザクション内で，同じブロックにアクセスできる関数（回復対象の DAM ファイルの場合）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
トランザクション内で，まだ DAM ファイルにアクセスする関数を呼び出していない	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read（参照目的の入力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read（参照目的の入力　排他を指定）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read（更新目的の入力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	○
	dc_dam_rewrite（更新の取り消し）	○
	dc_dam_write（出力）	DCDAMER_SEQER（01605）
dc_dam_rewrite（更新）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_dam_rewrite （更新）	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_rewrite （更新の取り消し）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_write （出力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○

（凡例）

○：エラーになりません。

表 4–2　異なるトランザクションで，同じブロックにアクセスできる関数（回復対象の DAM ファイルの場合）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
トランザクション内で，まだ DAM ファイルにアクセスする関数を呼び出していない	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read （参照目的の入力）	dc_dam_read（参照目的の入力）	○

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_dam_read （参照目的の入力）	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read （参照目的の入力　排他を指定）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	DCDAMER_EXCER（01602）※
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	DCDAMER_EXCER（01602）※
dc_dam_read （更新目的の入力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	DCDAMER_EXCER（01602）※
	dc_dam_read（更新目的の入力）	DCDAMER_EXCER（01602）※
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	DCDAMER_EXCER（01602）※
dc_dam_rewrite （更新）	dc_dam_read（参照目的の入力）	○
	dc_dam_read （参照目的の入力　排他を指定）	DCDAMER_EXCER（01602）※
	dc_dam_read（更新目的の入力）	DCDAMER_EXCER（01602）※
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	DCDAMER_EXCER（01602）※
dc_dam_rewrite （更新の取り消し）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_dam_rewrite （更新の取り消し）	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_write （出力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	DCDAMER_EXCER（01602）※
	dc_dam_read（更新目的の入力）	DCDAMER_EXCER（01602）※
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	DCDAMER_EXCER（01602）※

（凡例）

○：エラーになりません。

注※

flags に DCDAM_WAIT を設定した場合は，排他解除待ちになります。

(6)　DAM サービスの関数が DCDAMER_PROTO でエラーリターンする原因

DAM サービスの関数が DCDAMER_PROTO（COBOL 言語の場合は 01600）でエラーリターンする原因を次に示します。原因は呼び出した関数によって異なります。

1. dc_rpc_open 関数（COBOL 言語の場合は CBLDCRPC('OPEN ')）を呼び出していません。
2. 回復対象の DAM ファイルの場合，ユーザサービス定義の atomic_update の指定が'N'になっています。
3. 次に示すように，UAP を正しくリンケージしていません。
   - DAM サービスの API で TAM ファイルにアクセスする場合に使うライブラリ（-ltdam）を，不当にリンケージしています。
   - トランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。
4. ユーザサービス定義の atomic_update オペランドの指定が'N'の場合に，dc_dam_start 関数（COBOL 言語の場合は CBLDCDAM('STRT')）を呼び出していません（回復対象外の DAM ファイルの場合）。

(7)　DAM ファイルの状態の参照

使っている DAM ファイルの状態を，オンライン中に参照できます。DAM ファイルの状態を参照するときは，**dc_dam_status 関数【CBLDCDAM('STAT')】**を呼び出します。

dc_dam_status 関数で参照できる情報を次に示します。

- 論理ファイルのブロック数
- 論理ファイルのブロック長

- 論理ファイルに対応した物理ファイル名
- 論理ファイルの現在の状態（閉塞されているかどうか）
- DAM サービス定義で指定した論理ファイルの属性
- DAM サービス定義で指定した論理ファイルのセキュリティ属性

(a) dc_dam_status 関数を使う場合の注意

dc_dam_status 関数を呼び出すと，DAM サービスは情報を取得するための排他制御をします。そのため，dc_dam_status 関数を頻繁に使うと，排他の解除待ちが起こってスループットが低下することがあります。オンライン中に DAM ファイルの状態を参照するのは，必要最小限にしてください。

4.1.5 オフライン中の DAM ファイルのアクセス（オフラインの業務をする UAP からの操作）

バッチ環境では DAM ファイルの内容を出力できます。オフラインでオープンした物理ファイルは，処理を終了したら必ずクローズさせてください。なお，オフラインで使用する関数は，オンライン（SUP，SPP，MHP）では使用できません。使用した場合の動作は保証できません。

(1) DAM ファイルにアクセスするときの名称

物理ファイルを，割り当てたときに設定した物理ファイル名を設定した dc_dam_iopen 関数【CBLDCDMB('OPEN')】でオープンします。物理ファイルをオープンしたときに**ファイル記述子**がリターンされます。ファイル記述子は，オープンしてからクローズするまでの処理で使うので，UAP 内で保持しておいてください。

オープン以降の処理（ブロックの入力，出力）は，このファイル記述子を関数に設定してアクセスします。物理ファイルのクローズもファイル記述子を設定した dc_dam_iclose 関数【CBLDCDMB('CLOS')】でクローズします。

(2) ブロックの入力と出力

dc_dam_iopen 関数でオープンした物理ファイルのブロックを入出力するときは，次に示す 2 種類の方法があります。

- **ファイルの先頭から順番に入出力する場合**
  ブロックを入力する場合は dc_dam_get 関数【CBLDCDMB('GET ')】，ブロックを出力する場合は dc_dam_put 関数【CBLDCDMB('PUT ')】を呼び出します。このとき，物理ファイルをオープンしたあとに，UAP の処理で入力と出力を混在できません。いったん物理ファイルをクローズしてからにしてください。
- **任意のブロックを入出力する場合**

任意のブロックを入出力する場合は，dc_dam_iopen 関数に OVERWRITE を設定して物理ファイルをオープンしてください。そのほかの関数を呼び出して物理ファイルをオープンした場合は，任意のブロックを入出力できません。

任意のブロックを入出力には，2 種類の方法があります。この 2 種類の方法は混在できません。どちらかの方法でブロックを入出力してください。

1. 入出力するブロックの相対ブロック番号を検索します。検索するときは，dc_dam_bseek 関数【CBLDCDMB('BSEK')】を呼び出します。

   検索し終わってからブロックを入力する場合は dc_dam_get 関数，ブロックを出力する場合は dc_dam_put 関数を使います。このとき，dc_dam_iopen 関数から dc_dam_iclose 関数の処理の間では，入力と出力を混在できます。

2. 入出力するブロックの相対ブロック番号を指定して，次のどちらかの関数を呼び出します。入力と出力は混在できます。

- 入力する場合：dc_dam_dget 関数【CBLDCDMB('DGET')】
- 出力する場合：dc_dam_dput 関数【CBLDCDMB('DPUT')】

複数ブロックを一括入出力する指定は，ファイルの先頭から順番に入出力する場合だけ有効です。

(3) 複数ブロックの一括入出力

物理ファイルを割り当てるときとオープンするときに，入出力する単位としてブロック数を指定できます。

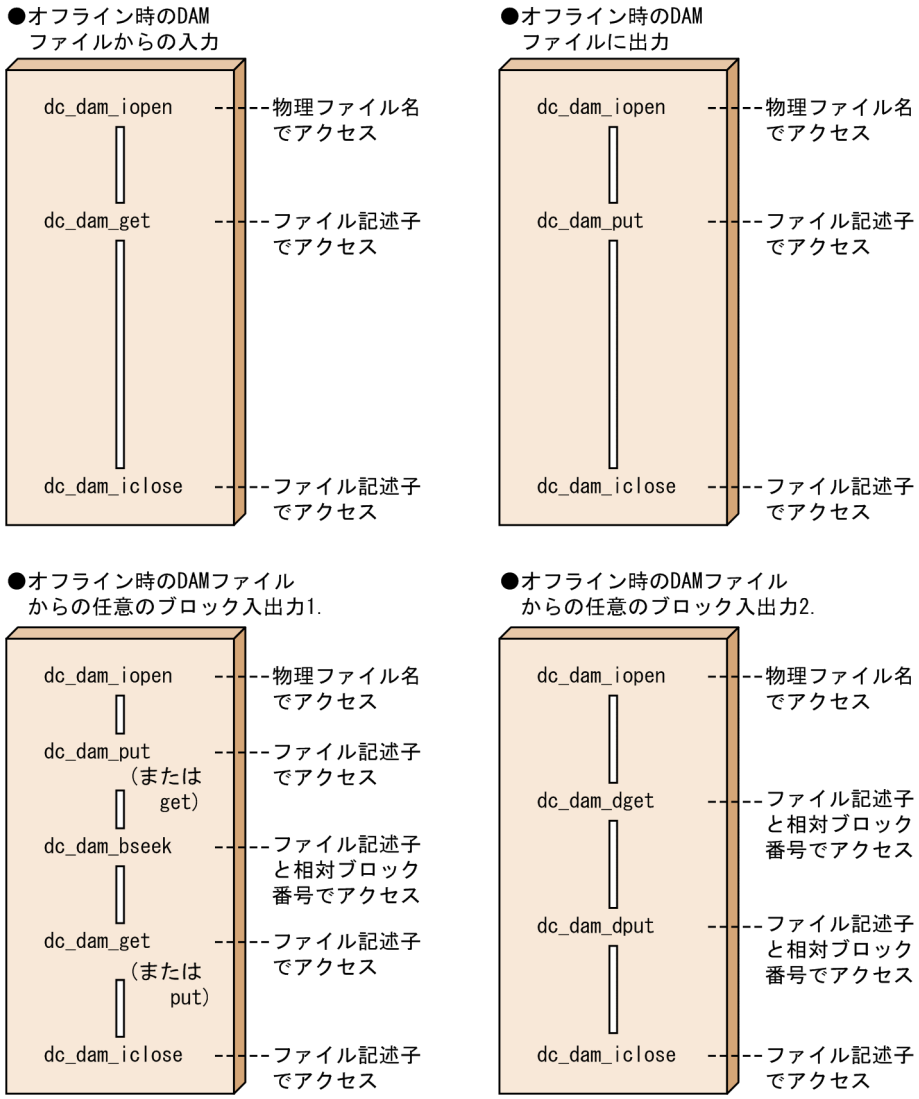
(4) ブロックの初期作成と再作成

入力したブロックを物理ファイルに出力するときに，出力するブロック以降にある残りの領域を，ヌル文字で埋めるかどうかを設定できます。出力するブロック以降の残りの領域をヌル文字で埋めるときは INITIALIZE を設定します。ヌル文字でクリアしないときは，OVERWRITE を設定します。OVERWRITE を設定した場合には，任意のブロックの入出力ができます。このとき，出力したブロック以降の領域は更新されないでそのまま残ります。

残りのブロックをヌル文字で埋めるかどうかは，dc_dam_iopen 関数の引数で指定します。この指定は，dc_dam_put 関数でブロックを出力して dc_dam_iclose 関数でクローズしたときに有効です。dc_dam_iclose 関数を呼び出さないで UAP を終了した場合は，残りのブロックをヌル文字で埋めません。ヌル文字で埋める場合は，必ず dc_dam_iclose 関数を呼び出してください。

オフライン中の DAM ファイルのアクセス手順を次の図に示します。

図 4-3　オフライン時の DAM ファイルのアクセス手順



4.1.6　物理ファイルの作成（オフラインの業務をする UAP からの操作）

物理ファイルはオフラインで作成します。物理ファイルは，まず OpenTP1 ファイルシステムに dc_dam_create 関数【CBLDCDMB('CRAT')】で割り当てます。割り当てるときに設定する内容を次に示します。

- 割り当てる物理ファイル名
- 一つのブロックのブロック長と，ブロック数
- 入出力の単位になる，一括処理ブロック数
- ファイル所有者，所有者グループ，ほかの UAP からのアクセス権

物理ファイルを割り当てたときに，**ファイル記述子**がリターンされます。

ファイル記述子は，オープン以降の処理で使います。dc_dam_create 関数で返されたファイル記述子を使える関数を次に示します。

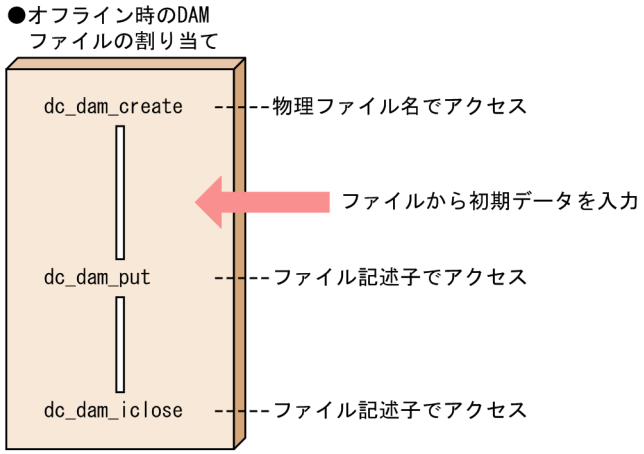
- dc_dam_put 関数（ブロックの出力）
- dc_dam_iclose 関数（ファイルのクローズ）

次に示す関数では，dc_dam_create 関数で返されたファイル記述子を使えません。関数はエラーリターンします。

- dc_dam_get 関数（ブロックの入力）
- dc_dam_dget 関数，dc_dam_dput 関数（ブロックの直接入出力）
- dc_dam_bseek 関数（ブロックの検索）

OpenTP1 ファイルシステムに，最初に物理ファイルとして DAM ファイルを作成する手順を次の図に示します。

図 4-4　DAM ファイルの作成手順



4.1.7　DAM ファイルの排他制御

DAM ファイルの更新中に，ほかの UAP からの更新処理が割り込むと，一つの論理ファイルに二つの処理が同時に反映されて，ファイルの内容に矛盾が生じてしまいます。このようなことを防ぐため，DAM ファイルにアクセスする関数で**排他制御**をします。排他を制御することで，複数の UAP からアクセスされる DAM ファイルでも，データの整合性を保証できます。

- **DAM ファイルへアクセスするトランザクションの範囲（回復対象の DAM ファイルの場合）**

  DAM ファイルへアクセスするときは，**トランザクションブランチ単位**の排他制御となります。そのため一つのグローバルトランザクションに属する複数のトランザクションブランチから同じブロックまたは同じファイルにアクセスしても，排他エラーになる場合があります。このようなエラーを避けるため，**グローバルトランザクション単位**で排他制御することもできます。グローバルトランザクション単

位で排他制御する場合は，該当する DAM ファイルの DAM サービス定義にグローバルトランザクション単位でアクセスすることを指定します。

グローバルトランザクション単位で排他制御する場合，トランザクションブランチごとに DAM ファイルにアクセスしても，並列にアクセスできないで順次アクセスになります。そのため，トランザクションの性能が下がる場合があります。トランザクションブランチごとの DAM ファイルへのアクセスを並列に処理させる場合は，グローバルトランザクション単位での排他制御は指定しないでください。

(1) 排他制御モード

DAM ファイルアクセス時の排他の条件を**排他制御モード**といいます。排他制御モードには，次の 2 種類があります。

参照目的の排他（共用モード　PR　Protected Retrieve）

UAP は排他指定したファイルの参照だけできます。ほかのトランザクションには，参照だけを許可します。

更新目的の排他（排他モード　EX　EXclusive）

UAP は排他指定したファイルの参照，更新ができます。ほかのトランザクションには，参照，更新を禁止します。

(2) 排他の指定単位（回復対象の DAM ファイルの場合）

オンライン中の DAM ファイルへのアクセスに対する排他の指定単位には，次の 2 種類があります。

(a) ブロック排他

ブロック単位に排他制御をします。ブロックを参照するときは共用モードの排他が掛かり，ブロック更新，またはブロック出力時には排他モードの排他が掛かります。参照目的での排他指定は，オプションの指定で排他をしない（ほかの UAP に参照／更新を許す）こともできます。確保された排他の指定は，DAM ファイルへの処理を指定したトランザクション処理が正常終了したときに解除されます。

(b) ファイル排他

論理ファイル単位で排他制御します。論理ファイル全体に対して，ファイルのオープンからトランザクション処理の正常終了まで排他が掛かります。

ファイル排他を指定できるのは次の場合です。

- トランザクションブランチ単位で排他制御する指定で，トランザクションの範囲内で論理ファイルをオープンした場合

次に示す場合は，ファイル排他を指定できません。ブロック排他で排他制御をしてください。

- トランザクションの範囲外で論理ファイルをオープンした場合
- グローバルトランザクション単位で排他制御する指定をした場合

(3) 資源の排他解除待ちの設定（回復対象の DAM ファイルの場合）

- dc_dam_open 関数

  入出力しようとしたブロックが，すでにほかのトランザクションから排他をかけられていた場合（**排他エラー**），アクセスした関数をエラーリターンするか排他が解除されるのを待つかを，DAM ファイルのオープン時に dc_dam_open 関数の引数で設定できます。

  ファイル排他で DAM ファイルをオープンする場合，排他待ち種別は設定できません。

  dc_dam_open 関数でファイルをオープンしようとしたときに排他エラーが起こった場合は，無条件にエラーリターンします。

- dc_dam_read 関数と dc_dam_write 関数

  排他エラーが起こった場合，dc_dam_read 関数と dc_dam_write 関数の排他待ち種別（エラーリターンするか，排他が解除されるのを待つか）を設定できます。この設定を省略した場合は，dc_dam_open 関数に設定した値に従います。

  排他が解除されるのを待つと設定した場合に，デッドロックやタイムアウトが起こったときは，排他の解除を待っている関数がエラーリターンしたあと，**デッドロック情報**が出力されます。デッドロックやタイムアウトで関数がエラーリターンした場合は，トランザクションの同期点を取得して，確保した資源をすべて解放してください。

(4) オンライン時とオフライン時の排他

オンラインで使っている DAM ファイルは，オフラインでアクセスできません。オンラインで使っている DAM ファイルにオフラインのアクセスをする場合は，damhold，damrm コマンドで，いったんオンラインから削除してからオフラインでアクセスしてください。その後，damadd コマンドでオンラインに復帰させてください。

オフライン時でも，異なる UAP が同じ DAM ファイルに同時にはアクセスできません。DAM ファイルをオープンした UAP のプロセスが，クローズするまでその DAM ファイルを占有します。

4.1.8 回復対象外の DAM ファイルへのアクセス

トランザクションによる整合性の管理や障害時の回復を保証しない DAM ファイルを作成できます。このような DAM ファイルを**回復対象外の DAM ファイル**といいます。回復対象外の DAM ファイルのブロックは，トランザクションの処理でなくても，dc_dam_write 関数，dc_dam_rewrite 関数で更新できます。

(1) 回復対象外の DAM ファイルの定義

回復対象外にする DAM ファイルを定義するときには，DAM サービス定義の damfile 定義コマンドに-n オプションを付けて指定します。

(2)　回復対象外の DAM ファイルへのアクセス

ファイルへアクセスをする前に dc_dam_start 関数【CBLDCDAM('STRT')】を，アクセスを完了するときには dc_dam_end 関数【CBLDCDAM('END ')】を呼び出します。dc_dam_start 関数を呼び出したときは，アクセス終了後，必ず dc_dam_end 関数を呼び出してください。

回復対象外の DAM ファイルへは，DAM サービスの関数を使ってアクセスします。回復対象の DAM ファイルへアクセスする場合と同様に，アクセスできます。

ファイルをオープンしたときのファイル記述子は，次に示すことが起こるまで有効です。

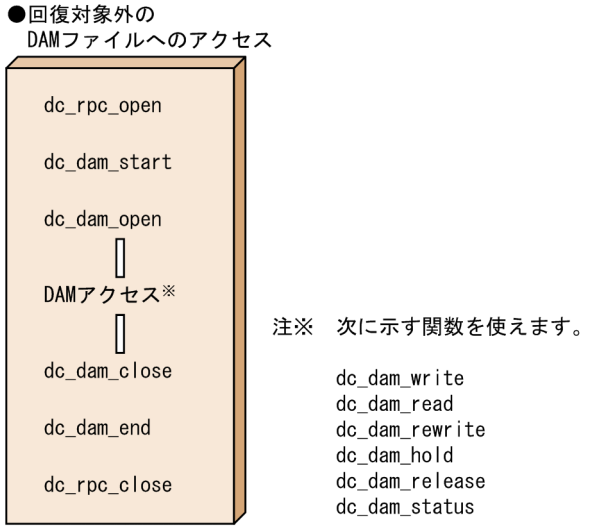
- 論理ファイルのクローズ
- 回復対象外の DAM ファイルの使用終了（dc_dam_end 関数の呼び出し）
- UAP のプロセスの終了

(3)　ファイルアクセスでエラーが起こった場合の処置

回復対象外の DAM ファイルへのアクセスでエラーが起こった場合でも，ファイルデータの障害は回復できません。

回復対象外の DAM ファイルへのアクセス手順を次の図に示します。

図 4-5　回復対象外の DAM ファイルへのアクセス手順



(4)　回復対象外の DAM ファイルへアクセスする関数の関係

以前にアクセスした関数によっては，同じ UAP の処理からでも，一つのブロックに対するアクセスがエラーリターンすることがあります。エラーリターンするかどうかは，一つの UAP から呼び出したときと異なる UAP から呼び出したときで異なります。以前に呼び出した関数と DAM ファイルのブロックにアクセスできる関数を表 4-3 と表 4-4 に示します。

表 4-3　一つの UAP 内で，同じブロックにアクセスできる関数（回復対象外の DAM ファイルの場合）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_dam_start 関数を呼び出したあと，まだ DAM ファイルにアクセスする関数を呼び出していない	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read （参照目的の入力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read （参照目的の入力　排他を指定）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read （更新目的の入力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	○
	dc_dam_rewrite（更新の取り消し）	○
	dc_dam_write（出力）	○
dc_dam_rewrite （更新）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_dam_rewrite （更新）	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_rewrite （更新の取り消し）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_write （出力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○

（凡例）

○：エラーになりません。

表 4-4　異なる UAP 内で，同じブロックにアクセスできる関数（回復対象外の DAM ファイルの場合）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_dam_start 関数を呼び出したあと，まだ DAM ファイルにアクセスする関数を呼び出していない	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read （参照目的の入力）	dc_dam_read（参照目的の入力）	○

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_dam_read （参照目的の入力）	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read （参照目的の入力　排他を指定）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_read （更新目的の入力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	DCDAMER_EXCER（01602）※
	dc_dam_read（更新目的の入力）	DCDAMER_EXCER（01602）※
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	DCDAMER_EXCER（01602）※
dc_dam_rewrite （更新）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_rewrite （更新の取り消し）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_dam_rewrite （更新の取り消し）	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○
dc_dam_write （出力）	dc_dam_read（参照目的の入力）	○
	dc_dam_read（参照目的の入力　排他を指定）	○
	dc_dam_read（更新目的の入力）	○
	dc_dam_rewrite（更新）	DCDAMER_SEQER（01605）
	dc_dam_rewrite（更新の取り消し）	DCDAMER_SEQER（01605）
	dc_dam_write（出力）	○

（凡例）

○：エラーになりません。

注※

flags に DCDAM_WAIT を設定した場合は，排他解除待ちになります。

(5)　回復対象外の DAM ファイルの排他制御

回復対象の DAM ファイルと同様に，回復対象外 DAM ファイルでも排他制御をします。ここでは，回復対象外の DAM ファイルの排他制御について説明します。回復対象の DAM ファイルと回復対象外の DAM ファイルの比較については，「4.1.8(6)　回復対象の DAM ファイルと回復対象外の DAM ファイルとの比較」を参照してください。

(a)　回復対象外の DAM ファイルの排他範囲

回復対象外の DAM ファイルの場合，トランザクションの範囲内かどうかに関係なく，DAM ファイルへアクセスできます。

(b)　排他制御モード

回復対象外の DAM ファイルの排他制御モードは，回復対象の DAM ファイルと同じです。排他制御モードについては，「4.1.7(1)　排他制御モード」を参照してください。

(c)　排他の指定単位

回復対象外の DAM ファイルの排他の指定単位は，回復対象の DAM ファイルと同じです。排他の指定単位については，「4.1.7(2)　排他の指定単位（回復対象の DAM ファイルの場合）」を参照してください。

(d)　資源の排他解除待ちの設定

回復対象外の DAM ファイルの資源の排他解除待ちの設定は，次に示す点を除いて，回復対象の DAM ファイルと同じです。

- dc_dam_open 関数に設定する排他解除待ちの種別は，dc_dam_open 関数自体の指定を含みます。回復対象の DAM ファイルの場合は，dc_dam_open 関数で排他エラーが起こると無条件にエラーリターンしましたが，回復対象外の DAM ファイルでは，引数に設定した排他待ち種別に従って処理できます。

資源の排他待ちの設定については，「4.1.7(3)　資源の排他解除待ちの設定（回復対象の DAM ファイルの場合）」を参照してください。

(6)　回復対象の DAM ファイルと回復対象外の DAM ファイルとの比較

回復対象の DAM ファイルと回復対象外の DAM ファイルとの比較について説明します。アクセスの違いを表 4-5 に，アクセス時に排他制御する範囲の違いを表 4-6 に示します。

表 4-5　回復対象の DAM ファイルと回復対象外の DAM ファイルとのアクセスの違い

DAM サービスの関数	関数を呼び出す条件	DAM ファイルの種類とアクセスする位置		
		回復対象 DAM ファイル		回復対象外 DAM ファイル
		トランザクション範囲外	トランザクション範囲	
dc_dam_open	ファイル排他，排他解除待ち	×	○	○
	ファイル排他，即時リターン	×	○	○
	ブロック排他，排他解除待ち	○	○	○
	ブロック排他，即時リターン	○	○	○
dc_dam_close	トランザクションの範囲でファイルをオープン	○	○	○
	トランザクションの範囲外でファイルをオープン	－	×	○
dc_dam_read	参照目的の入力，排他なし	○	○	○
	参照目的の入力，排他を指定	×	○	○
	更新目的の入力，排他を指定	×	○	○
dc_dam_rewrite	更新目的の出力	×	○	○
	更新の取り消し	×	○	○
dc_dam_write	（条件なし）	×	○	○

（凡例）

○：該当する条件で，関数を呼び出せます。

×：該当する条件で，関数を呼び出すとエラーリターンします。

－：該当する条件では，関数を呼び出せません。

表 4-6　回復対象の DAM ファイルと回復対象外の DAM ファイルとのアクセス時に排他制御する範囲の違い

排他の指定単位※と呼び出す関数		排他制御モード	回復対象 DAM ファイルの場合	回復対象外 DAM ファイルの場合
ファイル排他	dc_dam_open	EX	• 同期点処理の終了まで排他	• dc_dam_close または dc_dam_end の理が終了するまで排他
ブロック排他	dc_dam_read（参照）	PR	• 同期点処理の終了まで排他	• dc_dam_read の処理が終了するまで排他
	dc_dam_read（更新）	EX	• 同期点処理の終了まで排他 • dc_dam_rewrite（取り消し）の処理が終了するまで排他	• dc_dam_rewrite（更新または取り消し）の処理が終了するまで排他
	dc_dam_write	EX	• 同期点処理の終了まで排他	• dc_dam_write の処理が終了するまで排他

注※

ここで示す排他の指定単位は，dc_dam_open 関数に設定した場合です。dc_dam_open 関数にファイル排他を指定した場合，dc_dam_read 関数，dc_dam_write 関数に設定した排他の指定単位は無効です。

4.1.9　DAM サービスと TAM サービスとの互換

DAM サービスの関数で，TAM ファイルのレコードにアクセスできます。詳細については，「4.2.9　TAM サービスと DAM サービスとの互換」を参照してください。

4.2 TAM ファイルサービス（TP1/FS/Table Access）

OpenTP1 専用のユーザファイルとして使える，**TAM ファイル**について説明します。TAM ファイルを使うと，直接編成ファイルへ高速にアクセスできます。

TAM ファイルを使う場合は，システムに TP1/FS/Table Access が組み込まれていることが前提となります。また，OpenTP1 の基本機能が TP1/Server Base の場合だけ使えます。TP1/LiNK では，TAM ファイルサービスは使えません。

4.2.1 TAM ファイルの構成

TAM ファイルを構成する一つ一つのデータを**レコード**といいます。TAM ファイルのキーとレコードは，メモリにロードされます。実際のファイルでなく，メモリへのキーアクセスによって，UAP からの高速アクセスを実現できます。このインデクス部とデータ部を **TAM テーブル**といいます。

TAM テーブルは，レコードに対応させるキーがある**インデクス部**と，レコードがある**データ部**から構成されます。

(1) TAM テーブルのインデクス種別

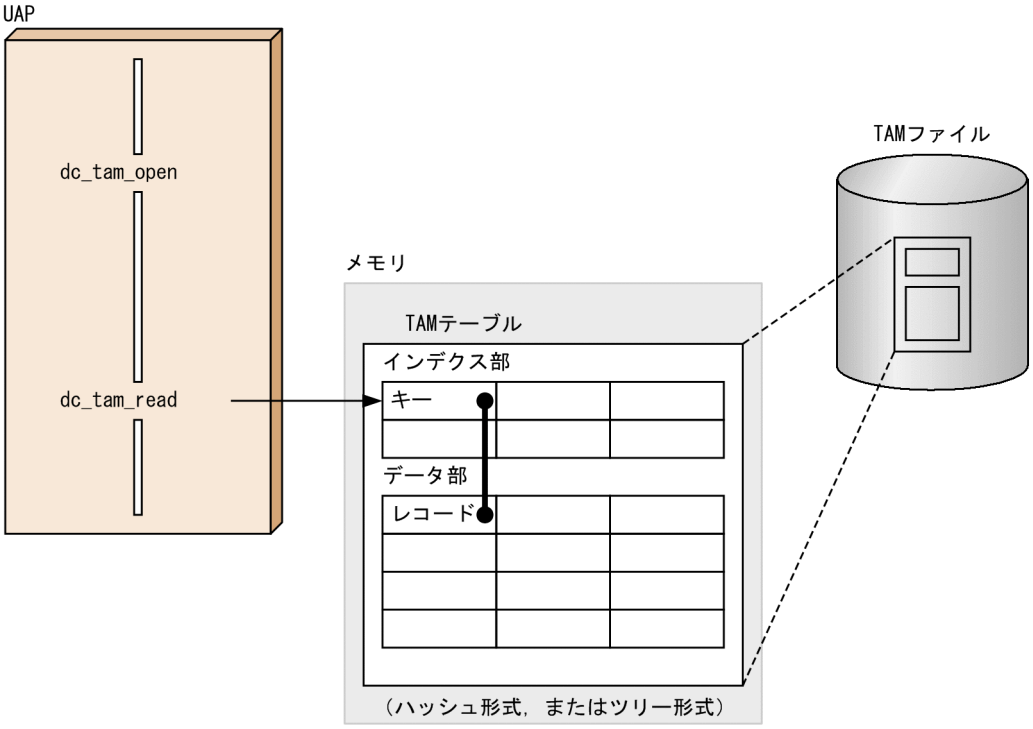
インデクス種別には，**ハッシュ形式**と**ツリー形式**があります。ハッシュ形式とツリー形式では，キーとレコードの対応のさせ方が異なります。また，TAM テーブルにアクセスするときは，使う TAM テーブルのインデクス種別を確認してから，UAP を作成してください。異なるインデクス種別の TAM テーブルにアクセスすると，UAP の TAM アクセス関数はエラーリターンします。

(2) TAM テーブルへのアクセス環境

TAM テーブルには，オンラインでだけアクセスできます。オフライン環境では TAM テーブルにはアクセスできません。また，UAP から TAM テーブルにアクセスするときは，TAM サービス定義で指定したアクセス形態に従ってください。定義されていないアクセスをすると，UAP の TAM アクセス関数はエラーリターンします。

TAM ファイルの構成を次の図に示します。

図 4-6　TAM ファイルの構成



4.2.2　TAM テーブルへアクセスするときの条件

TAM テーブルへのアクセスは，アクセスするトランザクションブランチが存在するノード（マシン）にある TAM ファイルのテーブルにだけできます。ノードごとの TAM テーブルに対する処理は独立しているので，TAM テーブル名はノードごとで管理しています。グローバルトランザクション内で TAM テーブルにアクセスするときは，ノード内でのテーブル名でアクセスしてください。

(1)　UAP から TAM テーブルへのアクセスとトランザクションの関数

TAM テーブルのオープンとクローズは，トランザクション開始前でも開始後でもできます。ただし，TAM テーブルのオープンとクローズ以外の関数（テーブルの参照や更新）は，必ずトランザクションを開始したあとで使ってください。

トランザクションを開始する前に TAM テーブルをオープンした場合は，オープン以降に開始したすべてのトランザクションを終了させてから，TAM テーブルをクローズしてください。

(2)　TAM テーブルへのアクセスと RPC の形態

TAM テーブルへのアクセスでは，グローバルトランザクションの RPC の形態がすべて**同期応答型 RPC** でなければなりません。非同期応答型 RPC，非応答型 RPC から TAM テーブルへアクセスした場合の動作は保証しません。

4.2.3　TAM テーブルへアクセスするときの名称

TAM テーブルは，**TAM テーブル名**でオープンします。TAM テーブルをオープンしたときに，そのテーブルを識別する名称として**テーブル記述子**がリターンされます。オープン以降の処理（レコードの入力，更新，追加，削除など）は，このテーブル記述子を関数に設定してアクセスします。

4.2.4　TAM テーブルへのアクセス手順

(1)　TAM テーブルのオープン

C 言語の UAP の場合，**dc_tam_open 関数**で TAM テーブルをオープンします※。TAM テーブルをオープンするときは，UAP ごとに dc_tam_open 関数を呼び出してください。

トランザクションの範囲内でも範囲外でも，TAM テーブルはオープンできます。ただし，トランザクションを開始する前に TAM テーブルをオープンする場合は，そのテーブルに対する排他はできません。TAM テーブルの排他については，「4.2.6　TAM テーブルの排他制御」を参照してください。

TAM テーブルのクローズもテーブル記述子を設定してクローズします※。テーブル記述子は，オープン以降の処理でも UAP 内で保持しておいてください。

注※

COBOL 言語で UAP をコーディングする場合は，UAP から TAM テーブルのオープンとクローズはしません。TAM テーブルにアクセスした時点で，該当の TAM テーブルがオープンされて，トランザクションが完了した時点で，該当の TAM テーブルはクローズされます。

(2)　レコードの入力／更新／追加／削除手順

TAM テーブルのレコードを参照または更新目的で入力するときは，**dc_tam_read 関数【CBLDCTAM('FxxR')('FxxU')('VxxR')('VxxU')】**を呼び出します。そのとき，別のグローバルトランザクションからの参照または更新を許すかどうかを設定できます。

TAM テーブルのレコードを更新するときは，dc_tam_read 関数でレコードを入力したあとに，**dc_tam_rewrite 関数【CBLDCTAM('MFY ')('MFYS')('STR ')('WFY ')('WFYS')('YTR ')】**を呼び出します（入力前提の更新）。

TAM テーブルからレコードを入力しないで，すでにあるレコードに上書きする更新，またはレコードを新規追加する場合は，**dc_tam_write 関数【CBLDCTAM('MFY ')('MFYS')('STR ')('WFY ')('WFYS')('YTR ')】**を呼び出します。

TAM テーブルのレコードを削除する場合は，**dc_tam_delete 関数【CBLDCTAM('ERS ')('ERSR')('BRS ')('BRSR')】**を呼び出します。削除するレコードは，任意のアドレスのバッファに退避できます。退避先のアドレスは，dc_tam_delete 関数に設定します。

dc_tam_read 関数，または dc_tam_delete 関数のバッファ域，および dc_tam_rewrite 関数，または dc_tam_write 関数のデータ域の先頭を，4 バイト境界で設定した場合，設定しない場合に比べて，より高速なアクセスができます。

(3) 複数レコードの一括入出力

複数のキー値（レコード）を，一括して入出力できます。TAM テーブルを入出力するときに，アクセスするキー値を構造体として関数に設定します。この構造体は複数個指定できます。

(4) インデクス種別によるレコードの入力

TAM テーブルからレコードを入力するときは，インデクス種別によって設定できる検索種別が異なります。

- ハッシュ形式の場合

  先頭検索と **NEXT 検索**ができます。これらの検索を使用すると，**全レコード検索**ができます。最初に dc_tam_read 関数（先頭レコード入力を設定）を呼び出して，先頭レコードを入力します。そのキー値以降のレコードは，dc_tam_read 関数の NEXT 検索で順に入力していきます。

  全レコード検索は，TAM テーブルのレコードを全件削除などに使用できます。TAM テーブルのレコードを全件削除する方法を次に示します。

  1. 先頭検索で先頭レコードを取得する。
  2. 先頭レコードをキー値に指定して NEXT 検索をして，次のレコードを取得する。
  3. 先頭レコードを削除する。
  4. 現在取得しているレコードをキー値に指定して NEXT 検索をして，次のレコードを取得する。
  5. 4 でキー値に指定したレコードを削除する。
  6. 次のレコードがなくなるまで 4，5 の手順を繰り返す。
  7. 次のレコードがなくなったら，最後の NEXT 検索でキー値に指定したレコードを削除する。

  この方法は，検索の開始位置としてキー値を指定するので，先頭から指定したキー値の前までの空のハッシュ域は検索しません。そのため，効率良く検索できます。

  次の方法は CPU を多く消費するため，レスポンスが遅延する可能性があります。

  1. 先頭検索で先頭レコードを取得する。
  2. 先頭レコードを削除する。
  3. レコードがなくなるまで 1，2 の手順を繰り返す。

  先頭検索はハッシュ域を先頭から探す処理です。この方法の場合，直前までの処理でレコードを削除して空になったハッシュ域を含めて，毎回先頭からレコードを検索するため，効率が悪く，レスポンスが遅延する可能性があります。

- ツリー形式の場合

設定したキー値に対して「=」,「<=」,「>=」,「<」，および「>」の条件で検索できます。検索後，キー値に該当するレコードを入力します。ある範囲の複数のキー値のレコードを入力するときは，「=」,「<=」,「>=」,「<」および「>」を設定すれば，その条件に該当するレコードを順に入力できます。

(5) TAM テーブルのクローズ

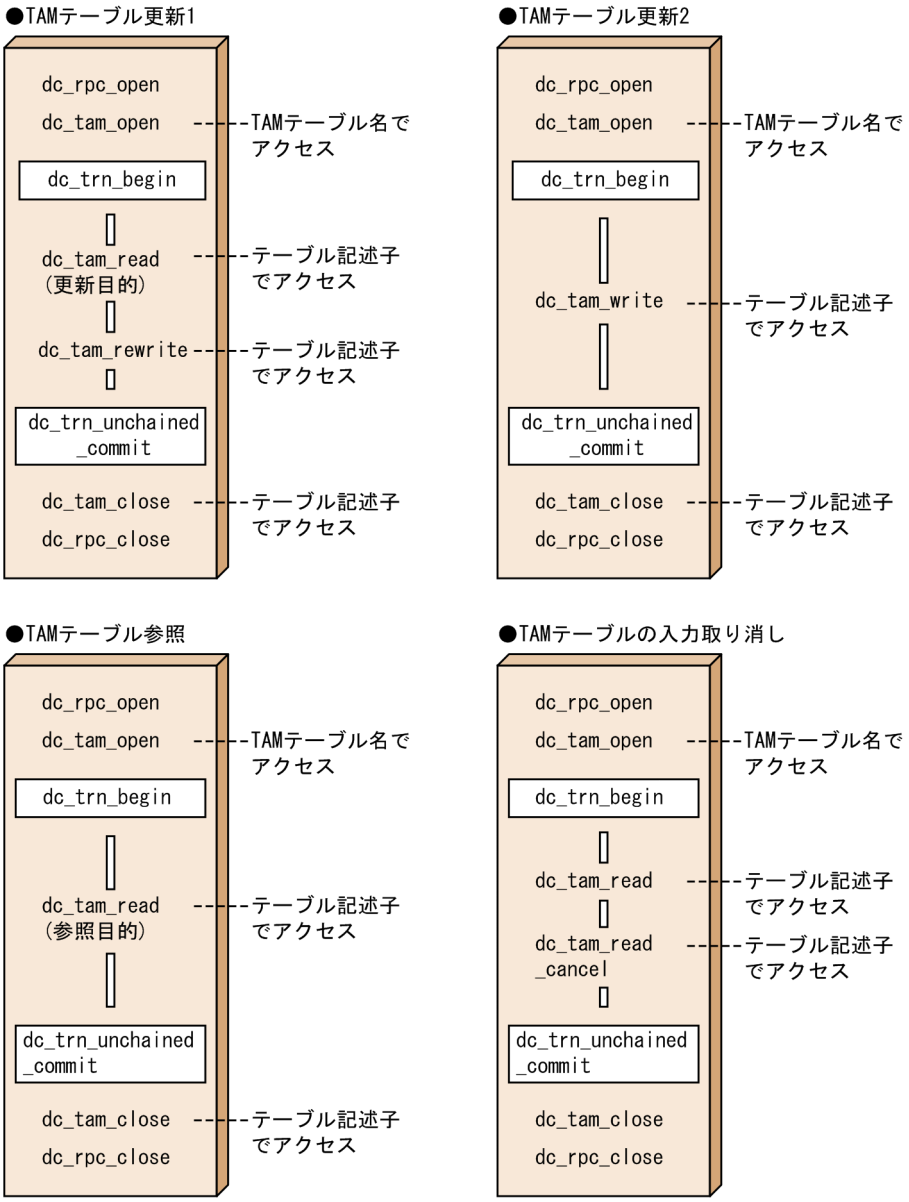
TAM テーブルは，dc_tam_close 関数でクローズします。

トランザクションの範囲内で TAM テーブルをオープンしたときは，トランザクション内でクローズしてください。クローズをしないでトランザクションを終了させたときは，OpenTP1 で TAM テーブルをクローズします。

トランザクションの範囲外で TAM テーブルをオープンしたときは，トランザクションの外でクローズしてください。トランザクション内で dc_tam_close 関数を呼び出すとエラーリターンします。

TAM テーブルのアクセス手順を次の図に示します。

図 4-7　TAM テーブルのアクセス手順



(6)　TAM テーブルの状態を取得

オンライン中に TAM テーブルの状態を知りたい場合は，dc_tam_get_inf 関数【CBLDCTAM('GST ')】を使います。dc_tam_get_inf 関数は，トランザクション処理からでもトランザクションでない処理からでも呼び出せます。dc_tam_get_inf 関数では，TAM テーブルが次に示すどの状態かを返します。

- オープン状態
- クローズ状態
- 論理閉塞状態
- 障害閉塞状態

dc_tam_get_inf 関数を呼び出した UAP が TAM テーブルをオープンしていなくても，ほかの UAP が TAM テーブルをオープンしていれば，dc_tam_get_inf 関数はオープン状態を返します。

(7) TAM テーブルの情報を取得

オンライン中に TAM テーブルの情報を知りたい場合は，dc_tam_status 関数【CBLDCTAM('INFO')】を使います。dc_tam_status 関数は，トランザクション処理からでもトランザクションでない処理からでも呼び出せます。dc_tam_status 関数は，次に示す TAM テーブルの情報を返します。

- TAM ファイル名
- TAM テーブルの状態
- 使用中のレコード数
- 最大レコード数
- インデクス種別
- アクセス形態
- ローディング契機
- TAM レコード長
- キー長
- キー開始位置
- セキュリティ属性

4.2.5 トランザクションと TAM アクセスの関係

トランザクションブランチで，TAM アクセスのエラーが起こった場合は，UAP から abort()を呼び出して，そのグローバルトランザクションのプロセスを異常終了させてください。

同じグローバルトランザクション内でも，以前にアクセスした関数によっては，一つのレコードに対するアクセスがエラーリターンする場合があります。また，同じレコードへのアクセスでも，同じグローバルトランザクションに属するときと，異なるグローバルトランザクションからの場合では，結果が異なります。

同じレコードに対して関数を複数回呼び出したときの処理結果を表 4-7 と表 4-8 に示します。

表 4-7 同じレコードに対して関数を複数回呼び出したときの処理結果（一つのグローバルトランザクション）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
トランザクション内で，まだ TAM テーブルにアクセスする関数を呼び出していない	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○
	dc_tam_read（更新目的の入力）	○
	dc_tam_read_cancel（入力の取り消し）	DCTAMER_SEQENCE（01732）
	dc_tam_rewrite（入力前提の更新）	DCTAMER_SEQENCE（01732）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
トランザクション内で，まだTAMテーブルにアクセスする関数を呼び出していない	dc_tam_write（更新）	○
	dc_tam_write（追加）	○
	dc_tam_delete（削除）	○
dc_tam_read （参照目的の入力）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○
	dc_tam_read（更新目的の入力）	○
	dc_tam_read_cancel（入力の取り消し）	DCTAMER_SEQENCE（01732）
	dc_tam_rewrite（入力前提の更新）	DCTAMER_SEQENCE（01732）
	dc_tam_write（更新）	○
	dc_tam_write（追加）	DCTAMER_EXKEY（01735）
	dc_tam_delete（削除）	○
dc_tam_read （参照目的の入力　排他を指定）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○
	dc_tam_read（更新目的の入力）	○
	dc_tam_read_cancel（入力の取り消し）	○※1
	dc_tam_rewrite（入力前提の更新）	DCTAMER_SEQENCE（01732）
	dc_tam_write（更新）	○
	dc_tam_write（追加）	DCTAMER_EXKEY（01735）
	dc_tam_delete（削除）	○
dc_tam_read （更新目的の入力）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○
	dc_tam_read（更新目的の入力）	○
	dc_tam_read_cancel（入力の取り消し）	○
	dc_tam_rewrite（入力前提の更新）	○
	dc_tam_write（更新）	○
	dc_tam_write（追加）	DCTAMER_EXKEY（01735）
	dc_tam_delete（削除）	○
dc_tam_read_cancel （入力の取り消し）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○
	dc_tam_read（更新目的の入力）	○
	dc_tam_read_cancel（入力の取り消し）	DCTAMER_SEQENCE（01732）※2

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_tam_read_cancel （入力の取り消し）	dc_tam_rewrite（入力前提の更新）	DCTAMER_SEQENCE（01732）
	dc_tam_write（更新）	○
	dc_tam_write（追加）	DCTAMER_EXKEY（01735）
	dc_tam_delete（削除）	○
dc_tam_rewrite （入力前提の更新）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○
	dc_tam_read（更新目的の入力）	○
	dc_tam_read_cancel（入力の取り消し）	DCTAMER_EXREWRT（01734）
	dc_tam_rewrite（入力前提の更新）	○
	dc_tam_write（更新）	○
	dc_tam_write（追加）	DCTAMER_EXKEY（01735）
	dc_tam_delete（削除）	○
dc_tam_write （更新，または追加）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○
	dc_tam_read（更新目的の入力）	○
	dc_tam_read_cancel（入力の取り消し）	DCTAMER_SEQENCE（01732）
	dc_tam_rewrite（入力前提の更新）	DCTAMER_SEQENCE（01732）
	dc_tam_write（更新）	○
	dc_tam_write（追加）	DCTAMER_EXKEY（01735）
	dc_tam_delete（削除）	○
dc_tam_delete （削除）	dc_tam_read（参照目的の入力）	DCTAMER_NOREC（01731）
	dc_tam_read（参照目的の入力　排他を指定）	DCTAMER_NOREC（01731）
	dc_tam_read（更新目的の入力）	DCTAMER_NOREC（01731）
	dc_tam_read_cancel（入力の取り消し）	DCTAMER_NOREC（01731）
	dc_tam_rewrite（入力前提の更新）	DCTAMER_NOREC（01731）※3
	dc_tam_write（更新）	DCTAMER_NOREC（01731）
	dc_tam_write（追加）	○
	dc_tam_delete（削除）	DCTAMER_NOREC（01731）

（凡例）

○：エラーになりません。

DCTAMER_NOREC（01731）：指定されたレコードはありません。

DCTAMER_SEQENCE（01732）：dc_tam_read 関数を呼び出していません。

DCTAMER_EXKEY（01735）：関数に設定したキー値のレコードがあるので，追加できません。

注※1

dc_tam_read 関数（参照目的の入力，排他を指定）の前に，dc_tam_rewrite 関数，dc_tam_write 関数を呼び出して，レコードを更新または追加されている場合は，DCTAMER_EXREWRT，または DCTAMER_EXWRITE がリターンされます。

注※2

dc_tam_read_cancel 関数（入力の取り消し）の前に dc_tam_rewrite 関数，dc_tam_write 関数を呼び出して，レコードを更新または追加されている場合は，DCTAMER_EXWRITE がリターンされます。

注※3

dc_tam_delete 関数（削除）の前に dc_tam_write 関数を呼び出して，レコードが追加されている場合は，DCTAMER_SEQENCE がリターンされます。

表 4-8　同じレコードに対して関数を複数回呼び出したときの処理結果（異なるグローバルトランザクション）

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
トランザクション内で，まだ TAM テーブルにアクセスする関数を呼び出していない	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○
	dc_tam_read（更新目的の入力）	○
	dc_tam_read_cancel（入力の取り消し）	－※1
	dc_tam_rewrite（入力前提の更新）	－※1
	dc_tam_write（更新）	○
	dc_tam_write（追加）	○
	dc_tam_delete（削除）	○
dc_tam_read（参照目的の入力）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○※2
	dc_tam_read（更新目的の入力）	○※2
	dc_tam_read_cancel（入力の取り消し）	－※1
	dc_tam_rewrite（入力前提の更新）	－※1
	dc_tam_write（更新）	○※2
	dc_tam_write（追加）	DCTAMER_EXKEY（01735）
	dc_tam_delete（削除）	○※2
dc_tam_read（参照目的の入力　排他を指定）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○※2
	dc_tam_read（更新目的の入力）	DCTAMER_LOCK（01736）※3
	dc_tam_read_cancel（入力の取り消し）	－※1

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_tam_read （参照目的の入力　排他を指定）	dc_tam_rewrite（入力前提の更新）	－※1
	dc_tam_write（更新）	DCTAMER_LOCK（01736）※3
	dc_tam_write（追加）	DCTAMER_LOCK（01736）※3
	dc_tam_delete（削除）	DCTAMER_LOCK（01736）※3
dc_tam_read （更新目的の入力）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	DCTAMER_LOCK（01736）※3
	dc_tam_read（更新目的の入力）	DCTAMER_LOCK（01736）※3
	dc_tam_read_cancel （入力の取り消し）	－※1
	dc_tam_rewrite（入力前提の更新）	－※1
	dc_tam_write（更新）	DCTAMER_LOCK（01736）※3
	dc_tam_write（追加）	DCTAMER_LOCK（01736）※3
	dc_tam_delete（削除）	DCTAMER_LOCK（01736）※3
dc_tam_read_cancel （入力の取り消し）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	○※4，※5
	dc_tam_read（更新目的の入力）	○※4，※5
	dc_tam_read_cancel （入力の取り消し）	－※1
	dc_tam_rewrite（入力前提の更新）	－※1
	dc_tam_write（更新）	○※4，※5
	dc_tam_write（追加）	DCTAMER_LOCK（01736）※3
	dc_tam_delete（削除）	DCTAMER_LOCK（01736）※3
dc_tam_rewrite （入力前提の更新）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	DCTAMER_LOCK（01736）※3
	dc_tam_read（更新目的の入力）	DCTAMER_LOCK（01736）※3
	dc_tam_read_cancel （入力の取り消し）	－※1
	dc_tam_rewrite（入力前提の更新）	－※1
	dc_tam_write（更新）	DCTAMER_LOCK（01736）※3
	dc_tam_write（追加）	DCTAMER_LOCK（01736）※3

前回呼び出した関数	今回呼び出した関数	結果またはエラーリターンする値
dc_tam_rewrite （入力前提の更新）	dc_tam_delete（削除）	DCTAMER_LOCK（01736）※3
dc_tam_write （更新，または追加）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	DCTAMER_LOCK（01736）※3
	dc_tam_read（更新目的の入力）	DCTAMER_LOCK（01736）※3
	dc_tam_read_cancel （入力の取り消し）	－※1
	dc_tam_rewrite（入力前提の更新）	－※1
	dc_tam_write（更新）	DCTAMER_LOCK（01736）※3
	dc_tam_write（追加）	DCTAMER_LOCK（01736）※3
	dc_tam_delete（削除）	DCTAMER_LOCK（01736）※3
dc_tam_delete （削除）	dc_tam_read（参照目的の入力）	○
	dc_tam_read（参照目的の入力　排他を指定）	DCTAMER_LOCK（01736）※3
	dc_tam_read（更新目的の入力）	DCTAMER_LOCK（01736）※3
	dc_tam_read_cancel（入力の取り消し）	－※1
	dc_tam_rewrite（入力前提の更新）	－※1
	dc_tam_write（更新）	DCTAMER_LOCK（01736）※3
	dc_tam_write（追加）	DCTAMER_LOCK（01736）※3
	dc_tam_delete（削除）	DCTAMER_LOCK（01736）※3

（凡例）

○：エラーになりません。

－：該当しません。

DCTAMER_EXKEY（01735）：関数に設定したキー値のレコードがあるので，追加できません。

DCTAMER_LOCK（01736）：排他エラーが起こりました。

注※1

異なるトランザクションでは，別の処理になります。

注※2

別グローバルトランザクションで，同じ TAM テーブルに対してレコードの追加／削除している場合は，DCTAMER_LOCK（01736）がリターンされます。ただし，排他待ち種別に DCTAM_WAIT を設定した場合は，排他解除待ちとなります。

注※3

排他待ち種別に DCTAM_WAIT を設定した場合は，排他待ちとなります。

注※4

別グローバルトランザクションでレコードの追加／削除している場合は，DCTAMER_LOCK（01736）がリターンされます。ただし，排他待ち種別に DCTAM_WAIT を設定した場合は，排他解除待ちとなります。

注※5

dc_tam_read_cancel 関数の前に，別グローバルトランザクションで，dc_tam_rewrite 関数，dc_tam_write 関数を呼び出して，レコードの更新または追加している場合は，DCTAMER_LOCK（01736）がリターンされます。ただし，排他待ち種別に DCTAM_WAIT を設定した場合は，排他解除待ちとなります。

4.2.6　TAM テーブルの排他制御

TAM テーブルの更新中に，ほかの UAP からの TAM テーブルを更新する処理が割り込むと，一つのレコードに二つの処理が同時に反映されて，テーブルの内容に矛盾が生じます。これを防ぐために TAM テーブルにアクセスする関数内に排他制御の指定をします。排他制御することで，複数の UAP からアクセスされる各データ間の整合性が保証されます。

TAM テーブルはグローバルトランザクション単位に排他制御します。

(1)　排他制御モード

TAM テーブルアクセス時の排他の条件を**排他制御モード**といいます。排他制御モードには次の 2 種類があります。

参照目的の排他（共用モード　PR　Protected Retrieve）

排他したレコードの参照だけできます。ほかのグローバルトランザクションからの参照だけを許可します。

更新目的の排他（排他モード　EX　EXclusive）

排他したレコードまたはテーブルの参照，更新ができます。ほかのグローバルトランザクションからの参照，更新を禁止します。

(2)　排他の指定単位

オンライン中の TAM テーブルへアクセスするときの排他の指定単位には，次の 2 種類があります。

(a)　レコード排他

レコード単位に排他制御します。レコードを参照目的で入力するときは，参照目的の排他をするか，または排他をしない（ほかの UAP に更新を許す）設定をします。更新目的の入力や更新では，更新目的の排他をします。確保された排他は，TAM テーブルへの処理を指定したトランザクションが終了したときに解除されます。

(b)　テーブル排他

テーブル単位に排他制御します。TAM テーブルをテーブル排他でオープン時，およびレコードの追加/削除をしたときに，TAM テーブル全体に対して更新目的の排他をします。確保された排他は，TAM テーブルへの処理を指定したトランザクションが終了したときに解除されます。トランザクションを開始する前にオープンした場合は，テーブル排他はできません。

(3) 資源の排他解除待ちの設定

アクセスしようとしたTAMテーブルがすでにほかのUAPから排他を掛けられていた場合（排他エラー）に，アクセスした関数をエラーリターンするか，排他が解除されるのを待つかを，関数の引数に指定できます。

排他が解除されるのを待つと設定した場合に，デッドロックやタイムアウトが起こったときは，排他の解除を待っている関数がエラーリターンして，デッドロック情報が出力されます。デッドロックやタイムアウトで関数がエラーリターンした場合は，トランザクションの同期点を取得して，確保した資源をすべて解放してください。

COBOL言語で作成したUAPの場合，排他が解除されるのを待つかどうかを，次に示すどちらかで指定します。

- TAMサービス定義のtam_cbl_levelオペランド
- COBOL-UAP作成用プログラムCBLDCTAMのデータ名Iの設定

COBOL言語のUAPの場合に排他解除待ちを指定する方法については，マニュアル「OpenTP1 システム定義」および「OpenTP1 プログラム作成リファレンス COBOL言語編」を参照してください。

TAMサービスの関数の排他指定と実際に排他される状態を次の表に示します。

COBOL言語のUAPの場合は，レコードへアクセスするAPIで排他を確保します。

表 4-9　TAMサービスの関数の排他指定と実際に排他される状態

TAMサービスの関数とフラグに設定した値			TAMテーブルへの排他	TAMレコードへの排他
dc_tam_open関数	テーブル排他		更新排他	－※1
	レコード排他		レコードにアクセスする関数で排他を確保	
dc_tam_read関数 CBLDCTAM('FxxR'/'FxxU'/'VxxR'/'VxxU')	参照目的 ('FxxR'/'VxxR')	排他なし	－	－
		排他あり	参照排他※2	参照排他
	更新目的 (FxxU'/'VxxU')		参照排他※2	更新排他
dc_tam_rewrite関数			参照排他※3	更新排他※3
dc_tam_write関数 CBLDCTAM('MFY '/'MFYS'/'STR '/'WFY '/'WFYS'/'YTR ')	更新目的 ('MFY '/'WFY ')		参照排他※2	更新排他
	「更新または追加」('MFYS'/'WFYS')または「追加」('STR '/'YTR ')目的		更新排他	－※1
dc_tam_delete関数 CBLDCTAM('ERS '/'ERSR'/'ZRS '/'ZRSR')			更新排他	－※1

（凡例）

－：該当しません。

注※1

テーブル全体が更新排他で確保されるため，ほかのトランザクションからはアクセスできません。

注※2

「参照型」または「追加・削除できない更新型」のテーブルでは，TAM サービス定義でテーブル排他モードを「排他しない」とした場合は，この排他は確保されません。

注※3

更新目的の dc_tam_read 関数で，すでに資源は確保されています。

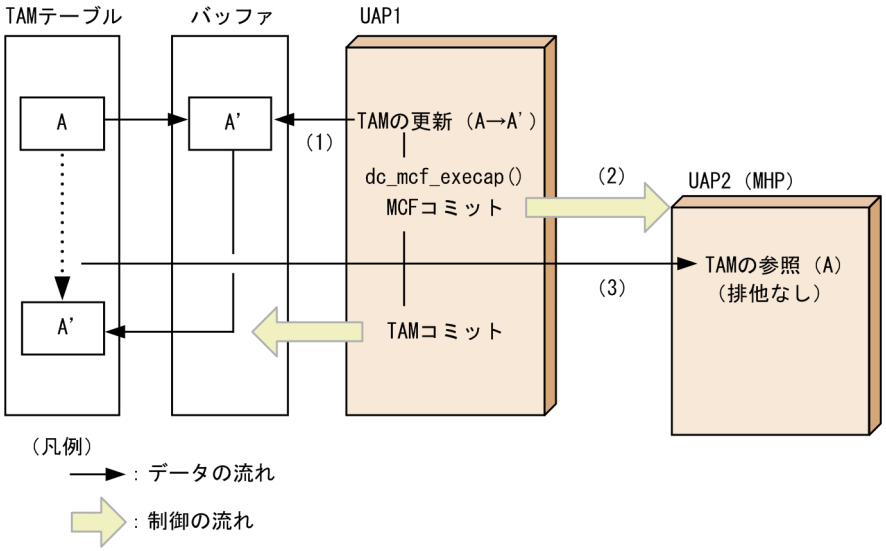
(4)　排他なし参照使用時の注意事項

レコード単位の排他制御ではレコードを参照目的で入力するときに排他をしない設定（排他なし参照）が選択できます。排他なし参照は，ほかの UAP からのアクセス状態に関係なく TAM アクセスができるので，性能を重視したい場合には有効な手段となります。ただし，使用に際しては特に次の点に注意してください。

- 排他制御を行わないので，同一データに対して参照と更新が競合する可能性のある業務には適しません。
- UAP からの更新結果が共用メモリ上の TAM テーブルに反映されるのは，更新を行った UAP のコミット完了の時点です。それまでは，ほかの UAP からは更新前のデータしか見ることができません。

  その一例を次に示します。

図 4-8　更新後の情報が排他なし参照時に反映されない例



(1)UAP1 より TAM の更新を行う

(2)UAP1 より dc_mcf_execap で UAP2（MHP）を起動する

(3)UAP2 から，(1)で更新した TAM の情報を参照する（排他なし参照）

この場合，(3)のタイミングでは(1)の更新結果が共用メモリ上の TAM テーブルに反映されている保証はありません。UAP2 が更新後の情報を期待しているようなアプリケーション設計を行うと処理不正となる可能性があります。

したがって，このような場合には，(3)では排他ありの参照を行う必要があります。

4.2.7　テーブル排他なし TAM テーブルアクセス機能

TP1/FS/Table Access　05-00 以前では，レコードを追加または削除する場合，テーブル単位の排他を確保します。これを**テーブル排他あり TAM テーブルアクセス機能**といいます。テーブル単位の排他については，「4.2.6　TAM テーブルの排他制御」を参照してください。

TP1/FS/Table Access　05-01 以降では，テーブル単位の排他を確保しないで，レコード単位の排他資源だけを確保して，TAM テーブルのレコードにアクセスできます。これを**テーブル排他なし TAM テーブルアクセス機能**といいます。

(1)　テーブル排他なし TAM テーブルアクセス機能の使用方法

テーブル排他なし TAM テーブルアクセス機能を使用するときは，TAM テーブルのアクセス形態に，「テーブル排他を確保しない，追加・削除できる更新型」を指定します。アクセス形態は，TAM サービス定義の tamtable コマンド定義句または tamadd コマンドで指定してください。tamtable コマンド定義句については，マニュアル「OpenTP1 システム定義」を，tamadd コマンドについてはマニュアル「OpenTP1 運用と操作」を参照してください。

同じ OpenTP1 システムで，テーブル排他なし TAM テーブルアクセス機能を使用する TAM テーブルと，テーブル排他あり TAM テーブルアクセス機能を使用する TAM テーブルを混在できます。

テーブル排他なし TAM テーブルアクセス機能を使用するときに，既存の TAM ファイルを tamcre コマンドによって再作成する必要はありません。

(2)　排他制御

(a)　排他の確保と解放

dc_tam_open 関数およびレコードにアクセスする関数（dc_tam_read，dc_tam_write，dc_tam_delete）で排他をします。COBOL 言語の UAP の場合は，レコードへアクセスするプログラムで排他をします。

確保された排他は，TAM テーブルにアクセスしたトランザクションが終了したときに解放されます。

(b)　テーブル排他なし TAM テーブルアクセス機能での TAM サービスの関数の排他指定と実際に排他される状態

テーブル排他なし TAM テーブルアクセス機能での TAM サービスの関数の排他指定と実際に排他される状態を次の表に示します。

表 4-10　テーブル排他なし TAM テーブルアクセス機能での TAM サービスの関数の排他指定と実際に排他される状態

TAM サービスの関数とフラグに指定した値			テーブル排他	レコード排他
dc_tam_open 関数	テーブル排他		更新排他※1	－
	レコード排他		－	－
dc_tam_read 関数	参照目的	排他なし	－	－
		排他あり	－	参照排他
	更新目的		－	更新排他
dc_tam_rewrite 関数			－	更新排他※2
dc_tam_write 関数			－	更新排他
dc_tam_delete 関数			－	更新排他

（凡例）

－：排他を確保しません。

注※1

テーブル排他指定の dc_tam_open 関数は，ほかのトランザクションのテーブル排他指定の dc_tam_open 関数とだけ排他で待ち合わせ，レコードにアクセスする関数とは待ち合わせをしません。詳細は，「4.2.7(3)　注意事項」を参照してください。

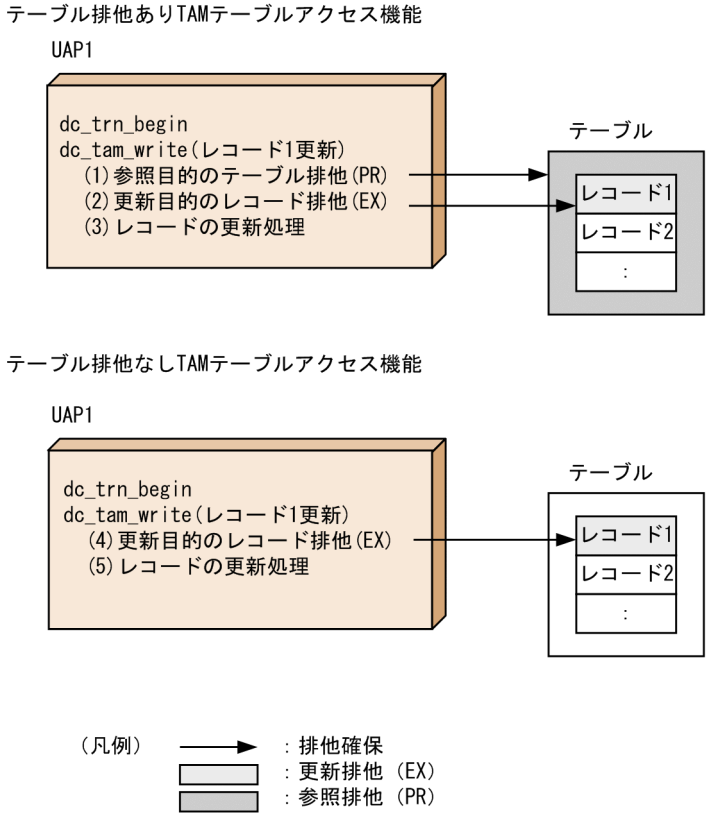
注※2

dc_tam_rewrite 関数では排他を確保しませんが，更新目的の dc_tam_read 関数で，すでに排他は確保されています。

(c)　排他の確保処理

テーブル排他あり TAM テーブルアクセス機能と，テーブル排他なし TAM テーブルアクセス機能の，レコード更新時の排他確保処理を次の図に示します。

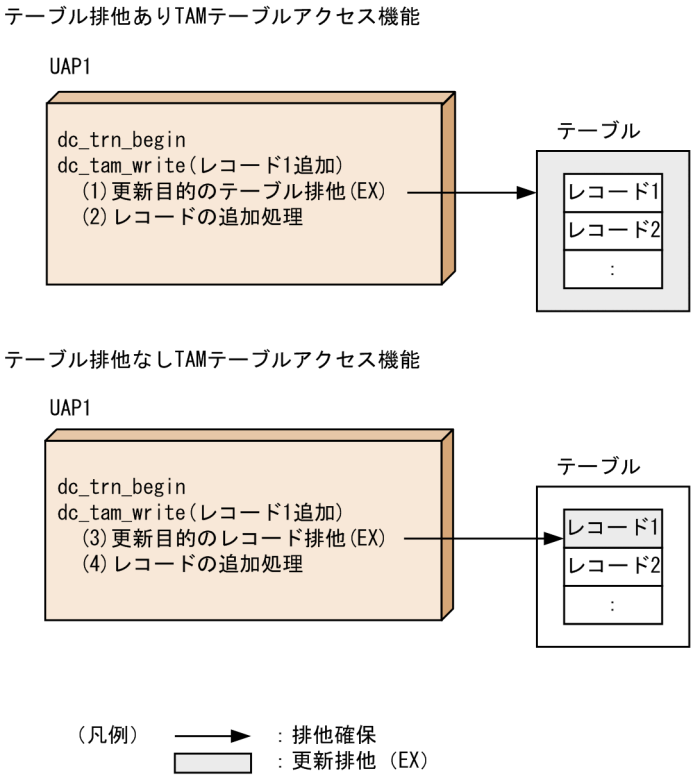
図 4-9　レコード更新時の排他確保処理



1. テーブル排他あり TAM テーブルアクセス機能では，dc_tam_write で，この図の(1)(2)(3)に示すように参照目的のテーブル排他を確保し，更新目的のレコード排他を確保して，レコードを更新します。
2. テーブル排他なし TAM テーブルアクセス機能では，dc_tam_write で，この図の(4)(5)に示すように更新目的のレコード排他を確保して，レコードを更新します。

テーブル排他あり TAM テーブルアクセス機能とテーブル排他なし TAM テーブルアクセス機能のレコード追加時の排他確保処理を次の図に示します。

図 4-10　レコード追加時の排他確保処理

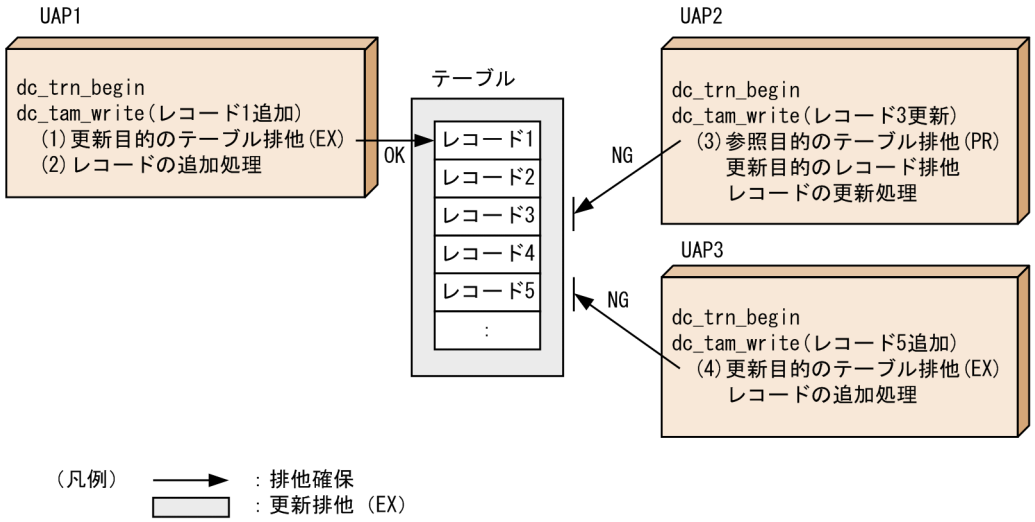


1. テーブル排他あり TAM テーブルアクセス機能では，dc_tam_write で，この図の(1)(2)に示すように更新目的のテーブル排他を確保して，レコードを追加します。
2. テーブル排他なし TAM テーブルアクセス機能では，dc_tam_write で，この図の(3)(4)に示すように，更新目的のレコード排他を確保して，レコードを追加します。

このように，テーブル排他なし TAM テーブルアクセス機能とテーブル排他あり TAM テーブルアクセス機能では排他の確保処理が異なるため，トランザクション間で TAM テーブルへのアクセスが競合した場合の動作も異なります。テーブル排他あり TAM テーブルアクセス機能では，レコードを追加または削除するトランザクションが存在すると，ほかのトランザクションからは同じ TAM テーブルに対してレコードを参照（排他あり），更新，追加，および削除できません。テーブル排他なし TAM テーブルアクセス機能では，アクセスするレコードが競合しなければ，同じ TAM テーブルにアクセスできます。

テーブル排他あり TAM テーブルアクセス機能で，レコードアクセスが競合した場合の処理を次の図に示します。

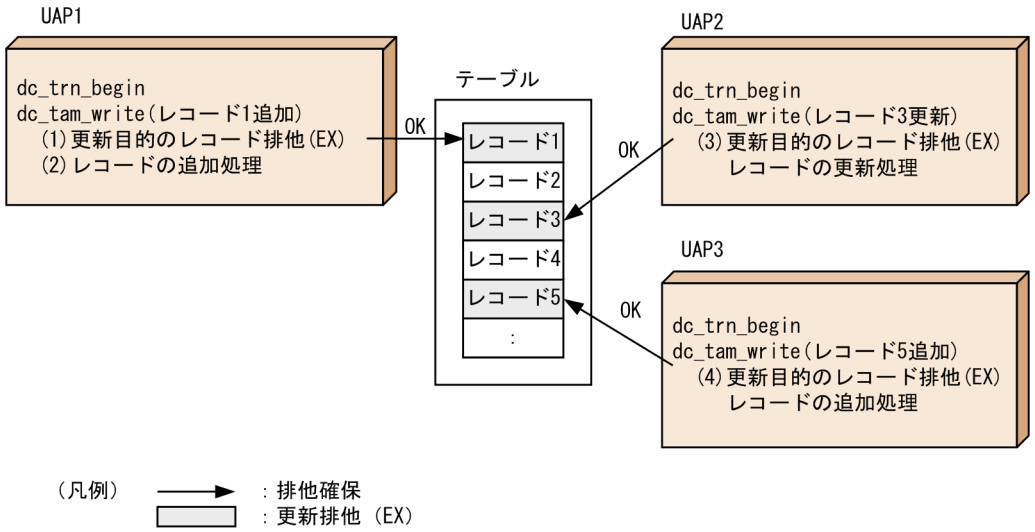
図 4-11　テーブル排他あり TAM テーブルアクセス機能でレコードアクセスが競合した場合の処理



1. UAP1 がレコード 1 を追加するとします。UAP1 では，この図の(1)(2)に示すように，更新目的のテーブル排他を確保して，レコードを追加します。
2. UAP2 では，この図の(3)に示すように，参照目的のテーブル排他を確保できないため，レコード 3 を更新できません。
3. UAP3 では，この図の(4)に示すように，更新目的のテーブル排他を確保できないため，レコード 5 を追加できません。
4. このため，UAP2 および UAP3 は，UAP1 のトランザクションが決着して排他が解放されるまで待つか，または DCTAMER_LOCK で異常終了します。

テーブル排他なし TAM テーブルアクセス機能で，レコードアクセスが競合した場合の処理を次の図に示します。

図 4-12　テーブル排他なし TAM テーブルアクセス機能でレコードアクセスが競合した場合の処理



1. UAP1 がレコード 1 を追加するとします。UAP1 では，この図の(1)(2)に示すように，レコード 1 に対して更新目的のレコード排他を確保して，レコードを追加します。

2. UAP2 では，この図の(3)に示すように，レコード 3 に対して更新目的のレコード排他を確保して，レコード 3 を更新します。
3. UAP3 では，この図の(4)に示すように，レコード 5 に対して更新目的のレコード排他を確保して，レコード 5 を追加します。
4. 以上のように，テーブル排他を確保しないため，UAP1 のトランザクションが決着していない状態でも，UAP2 および UAP3 は同じ TAM テーブルにアクセスできます。

(3) 注意事項

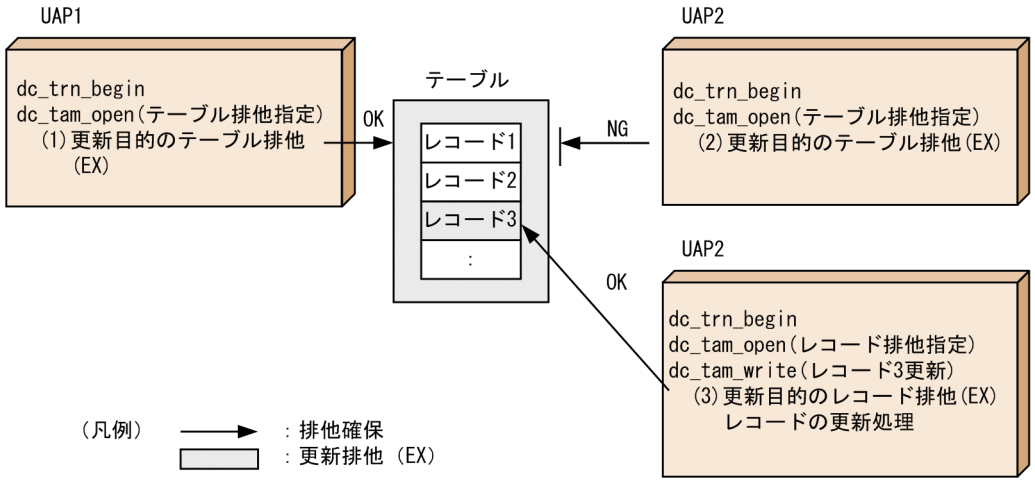
テーブル排他なし TAM テーブルアクセス機能を使用する場合，次の点に注意してください。

(a) dc_tam_open 関数のテーブル排他

dc_tam_open 関数をテーブル排他指定（flags に DCTAM_TBL_EXCLUSIVE を指定）で発行した場合，dc_tam_open 関数内でテーブル排他を確保しますが，レコードにアクセスする関数（dc_tam_read，dc_tam_write，dc_tam_delete）との排他制御はしません。つまり，テーブル排他指定の dc_tam_open 関数同士はテーブル排他で待ち合わせをしますが，テーブル排他指定の dc_tam_open 関数とレコードにアクセスする関数の間では排他の待ち合わせをしません。

dc_tam_open 関数の排他方式を次の図に示します。

図 4-13 dc_tam_open 関数の排他方式



1. UAP1 では，dc_tam_open 関数をテーブル排他指定（flags に DCTAM_TBL_EXCLUSIVE を指定）で発行し，この図の(1)に示す更新目的のテーブル排他を確保します。
2. UAP2 では，テーブル排他指定で dc_tam_open 関数を発行しますが，この図の(2)に示すように，更新目的のテーブル排他が確保できないため，UAP1 のトランザクションが決着して排他が解放されるまで待つか，または DCTAMER_LOCK で異常終了します。
3. UAP3 では，レコード排他指定（flags に DCTAM_REC_EXCLUSIVE を指定）で dc_tam_open 関数を発行しているため，dc_tam_open 関数は正常終了します。レコード 3 の更新では，この図の(3)に示すように，レコード 3 に対して更新目的のレコード排他を確保して，レコード 3 を更新します。

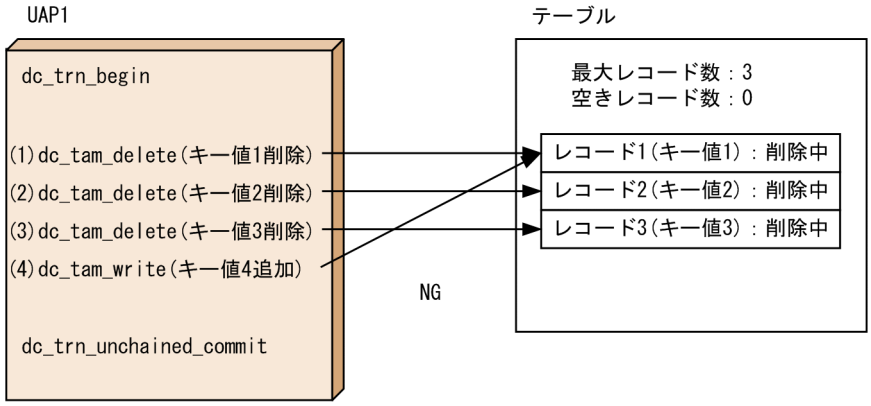
(b)　レコード追加時の空きレコードの割り当て

レコードの削除をした場合，レコードを削除したトランザクションがコミットするまで，削除したレコードは空きレコードにはなりません。そのため，レコードを削除したトランザクションがコミットするまで，削除したレコードの領域は追加するレコードに割り当てられません。ただし，レコードを削除したトランザクションと同一トランザクション内で，キー値が同じレコードを追加する場合には，削除したレコードの領域が割り当てられます。

したがって，追加するレコード数分の空きレコードがない状態でレコードを追加する場合，追加するトランザクションと同じトランザクションで異なるキー値のレコードを削除しても，レコードの追加はDCTAMER_NOAREA でエラーリターンします。

レコードの追加が DCTAMER_NOAREA となる例を次の図に示します。

図 4-14　レコードの追加が DCTAMER_NOAREA となる例

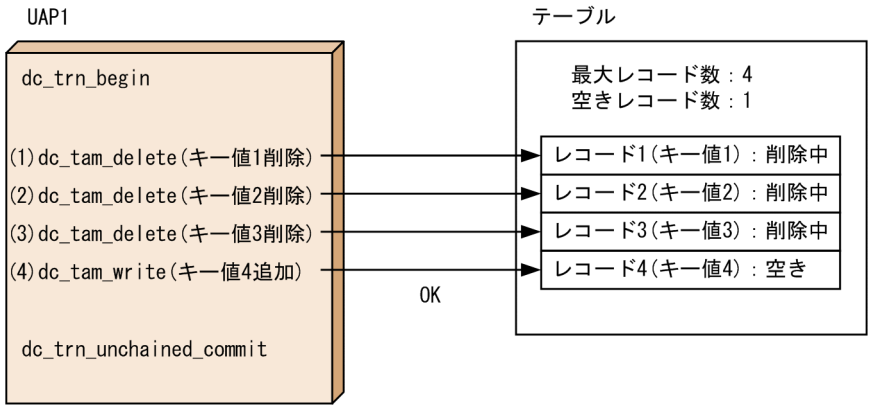


1. 最大レコード数が 3 の TAM テーブルにキー値 1，キー値 2，およびキー値 3 のレコードが格納されているとします。UAP1 では，キー値 1，キー値 2，キー値 3 を削除し，キー値 4 を追加します。
2. キー値 1 の削除では，この図の(1)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
3. キー値 2 の削除では，この図の(2)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
4. キー値 3 の削除では，この図の(3)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
5. キー値 4 の追加では，この図の(4)に示すように，空きレコードがないため，DCTAMER_NOAREA でエラーリターンします。

レコードの追加が DCTAMER_NOAREA とならないようにするには，追加するレコード数分の空きレコードを用意するか，またはレコードを削除したトランザクションをコミットしたあとでレコードを追加する必要があります。

追加するレコード数分の空きレコードを用意する場合の処理を次の図に示します。

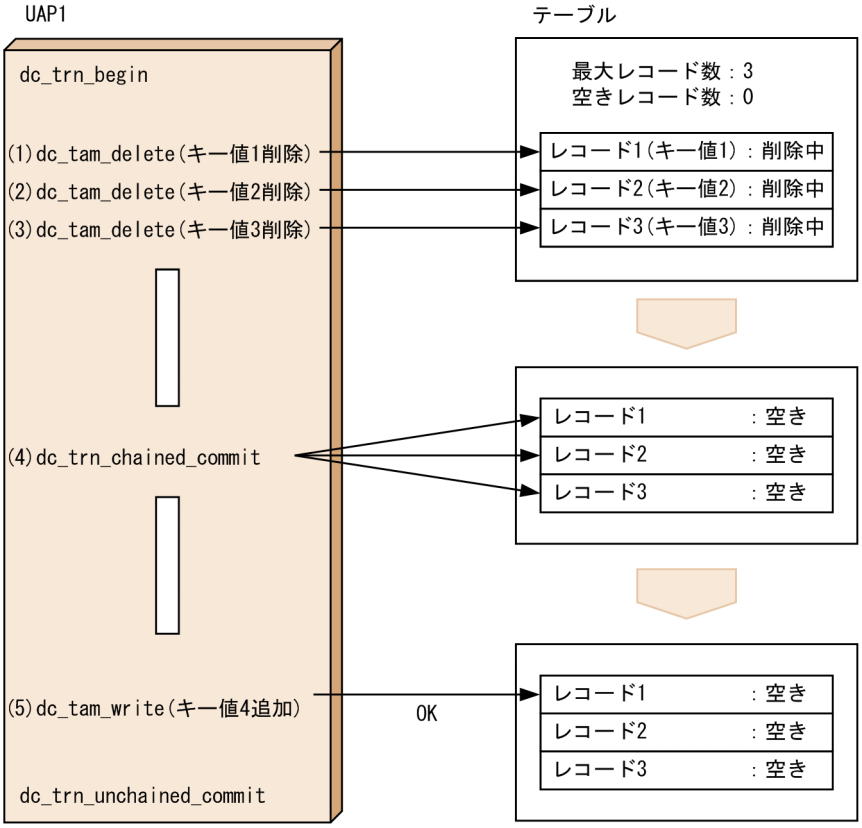
図 4-15　追加するレコード数分の空きレコードを用意する場合の処理



1. TAM テーブルの最大レコード数を 4 に増やします。TAM テーブルには，キー値 1，キー値 2，およびキー値 3 のレコードが格納されているとします。UAP1 では，キー値 1，キー値 2，キー値 3 を削除し，キー値 4 を追加します。
2. キー値 1 の削除では，この図の(1)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
3. キー値 2 の削除では，この図の(2)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
4. キー値 3 の削除では，この図の(3)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
5. キー値 4 の追加では，この図の(4)に示すように，空きレコードであるレコード 4 に追加できます。

レコードを削除したトランザクションをコミットしたあとでレコードを追加する場合の処理を次の図に示します。

図 4-16　レコードを削除したトランザクションをコミットしたあとでレコードを追加する場合の処理



1. 最大レコード数が 3 の TAM テーブルにキー値 1，キー値 2，およびキー値 3 のレコードが格納されているとします。UAP1 では，キー値 1，キー値 2，キー値 3 を削除したあとでいったんコミットし，次のトランザクションでキー値 4 を追加します。
2. キー値 1 の削除では，この図の(1)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
3. キー値 2 の削除では，この図の(2)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
4. キー値 3 の削除では，この図の(3)に示すように，レコード 1 は削除中になりますが，空きレコードにはなりません。
5. コミットでは，この図の(4)に示すように，削除中のレコード 1，レコード 2，レコード 3 を空きレコードにします。
6. キー値 4 の追加では，この図の(5)に示すように，空きレコードであるレコード 1 に追加できます。

(c)　アクセス形態の変更

tamadd コマンドによって，テーブル排他あり TAM テーブルアクセス機能を使用する TAM テーブルから，テーブル排他なし TAM テーブルアクセス機能を使用する TAM テーブルに変更したり，テーブル排他なし TAM テーブルアクセス機能を使用する TAM テーブルから，テーブル排他あり TAM テーブルア

クセス機能を使用する TAM テーブルに変更したりすることはできません。変更した場合，tamadd コマンドは異常終了します。

テーブル排他あり TAM テーブルアクセス機能を使用するか，テーブル排他なし TAM テーブルアクセス機能を使用するかを変更する場合は，TAM サービス定義の tamtable コマンド定義句を変更して OpenTP1 システムを正常開始するか，TAM サービス定義に登録しないで OpenTP1 システムを正常開始して tamadd コマンドで新規登録して変更してください。

(d)　デッドロック

テーブル排他あり TAM テーブルアクセス機能を使用していた TAM テーブルを，テーブル排他なし TAM テーブルアクセス機能を使用するように変更する場合，デッドロックとなることがあります。詳細は，「4.2.11(1)(b)　テーブル排他なし TAM テーブルアクセス機能を使用している場合」を参照してください。

(4)　プログラムインタフェース

dc_tam_status 関数および CBLDCTAM('INFO ')以外は，テーブル排他あり TAM テーブルアクセス機能と同じプログラムインタフェースを使用して TAM テーブルにアクセスできます。

なお，UAP は，リコンパイルまたは再リンケージが必要となることがあります。リコンパイルが必要な条件を表 4-11 に，再リンケージが必要な条件を表 4-12 に示します。

表 4-11　リコンパイルが必要な条件

条件				必要な作業
dc_tam_status 使用	あり	st_acs_type 参照	あり	アクセス形態の情報として，新定数 DCTAM_STS_RECLCK が返却されるので，UAP を修正し，リコンパイルし直す必要があります。
			なし	リコンパイルは不要です。
	なし			

表 4-12　再リンケージが必要な条件

条件		必要な作業
AP 使用ライブラリ	アーカイブライブラリ	再リンケージが必要です。
	共用ライブラリ	再リンケージは不要です。

dc_tam_status 関数では，DC_TAMSTAT 構造体の st_acs_type にアクセス形態を返却します。この st_acs_type に返却する値に DCTAM_STS_RECLCK を追加します。DCTAM_STS_RECLCK は，「テーブル排他を確保しない，追加・削除できる更新型」というアクセス形態を表し，テーブル排他なし TAM テーブルアクセス機能を使用する TAM テーブルであることを意味します。

dc_tam_status 関数のそのほかの返却値および使用方法については，マニュアル「OpenTP1 プログラム作成リファレンス C 言語編」を参照してください。

CBLDCTAM('INFO ')では，データ名 K にアクセス形態を返却します。このデータ名 K に返却する値に VALUE 'L'を追加します。VALUE 'L'は，「テーブル排他を確保しない，追加・削除できる更新型」というアクセス形態を表し，テーブル排他なし TAM テーブルアクセス機能を使用する TAM テーブルであることを意味します。CBLDCTAM('INFO ')のそのほかの返却値および使用方法については，マニュアル「OpenTP1 プログラム作成リファレンス COBOL 言語編」を参照してください。

(5) 定義インタフェース

TAM サービス定義の tamtable コマンド定義句では，-a オプションにアクセス形態を指定します。テーブル排他なし TAM テーブルアクセス機能を使用する場合，この-a オプションのオプション引数に reclck を指定してください。reclck は，「テーブル排他を確保しない，追加・削除できる更新型」というアクセス形態を表し，テーブル排他なし TAM テーブルアクセス機能を使用する TAM テーブルであることを意味します。

tamtable コマンド定義句のそのほかのオプションおよび使用方法については，マニュアル「OpenTP1 システム定義」を参照してください。

4.2.8 TAM ファイルの作成

OpenTP1 ファイルシステムに任意の直接編成ファイルを割り当てたあとに，tamcre コマンドで，TAM ファイルを作成します。このとき，インデクス種別，キー領域，レコードデータなどの指定をします。

4.2.9 TAM サービスと DAM サービスとの互換

(1) TAM テーブルにアクセスできる DAM サービスの関数

DAM ファイルサービスの関数（dc_dam_～）で，TAM テーブルのレコードにアクセスできます。この場合は，**DAM ファイルでの論理ファイル名を TAM テーブル名**として扱い，**DAM ファイルでの相対ブロック番号を TAM テーブルのキー値**として扱います。TAM テーブルにアクセスできる DAM サービスの関数は次のとおりです。

- dc_dam_open 関数（論理ファイルのオープン）
- dc_dam_close 関数（論理ファイルのクローズ）
- dc_dam_read 関数（論理ファイルのブロックの入力）
- dc_dam_rewrite 関数（論理ファイルのブロックの更新）
- dc_dam_write 関数（論理ファイルのブロックの出力）

dc_dam_hold 関数（論理ファイルの閉塞），dc_dam_release 関数（論理ファイルの閉塞解除）を TAM テーブルに対して呼び出した場合は正常にリターンしますが，実際には閉塞／閉塞解除されません。

DAM サービスの関数のうち，TAM ファイルのレコードにアクセスできない関数を次に示します。

- オフラインの業務で使う関数すべて
- dc_dam_start 関数（回復対象外 DAM ファイルの使用の開始）
- dc_dam_end 関数（回復対象外 DAM ファイルの使用の終了）
- dc_dam_status 関数（論理ファイルの状態の参照）

(2) DAM ファイルのデータを読み込んで，TAM アクセスできるようにする方法

DAM ファイルを TAM アクセスできるようにするには，次の手順でファイルを変更します。

1. dc_dam_get 関数で DAM ファイルのデータを入力して，各データにキー値を付けたあとに，任意のファイルに格納する。
2. 1.のファイルを入力として，tamcre コマンドを実行して TAM ファイルを作成する。

4.2.10 TAM サービスの統計情報

TAM サービスで取得するトランザクション単位の統計情報は，グローバルトランザクション単位に取得します。したがって，統計情報を出力するかどうかは，ルートトランザクションブランチのユーザサービス定義に指定した値に従います。

4.2.11 TAM のレコード追加・削除に伴う注意事項

排他確保によるデッドロックを発生させないためには，各 UAP で確保する資源に対する排他の種類・順序を整理する必要があります。ここでは，TAM レコードの追加・削除に伴う排他確保によるデッドロックの発生と注意について説明します。

(1) トランザクションのデッドロック要因

(a) テーブル排他あり TAM テーブルアクセス機能を使用している場合

「追加削除できる更新型」の TAM テーブルで，同一トランザクション内で同一テーブルに対して「追加(削除)，およびその他 TAM アクセス」を行う場合は，このトランザクションがデッドロックの要因になる可能性があります。これは次の要因を満たす場合です。

- 同一トランザクション内で同一テーブルに対して「追加（削除)，およびその他 TAM アクセス（排他あり)」を行う。
- 「追加（削除)，およびその他 TAM アクセス」のアクセス順序が次のようになった場合
  「その他 TAM アクセス」→「追加（削除)」

- TAM テーブルのオープンが次のどちらかの場合
  - トランザクション外
  - トランザクション内でレコード排他（COBOL 言語の場合はこのタイプ）
- 上記トランザクションと同時に，排他確保の必要がある別トランザクションが同一テーブルにアクセスする。

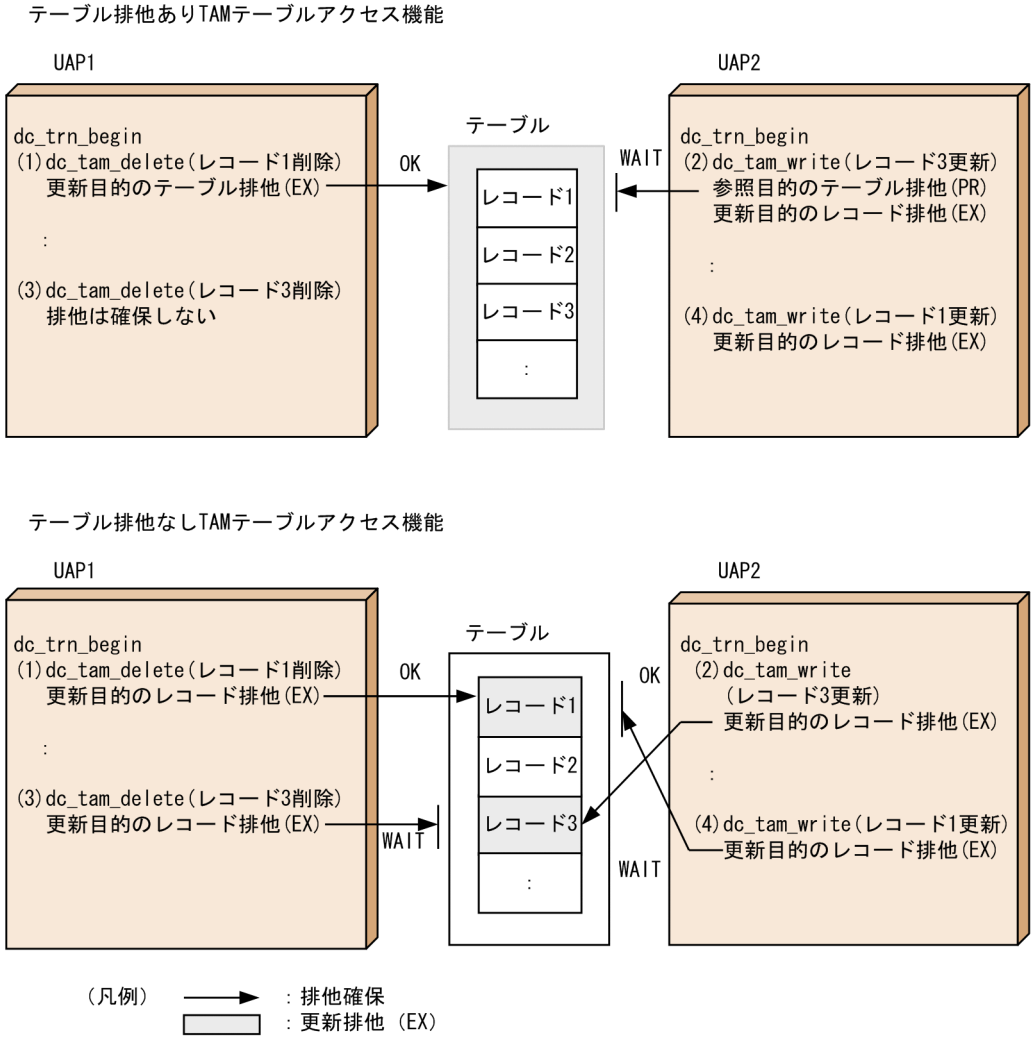
(b) テーブル排他なし TAM テーブルアクセス機能を使用している場合

テーブル排他あり TAM テーブルアクセス機能でレコードを更新，追加，または削除していた TAM テーブルを，テーブル排他なし TAM テーブルアクセス機能を使用するように変更する場合，デッドロックとなることがあります。

テーブル排他あり TAM テーブルアクセス機能でレコードを更新，追加，または削除する場合，更新排他でテーブル排他を確保します。このため，追加または削除するレコードの順序が同じレコードにアクセスするほかのトランザクションと異なっていても，テーブル排他で待ち合わせているのでデッドロックにはなりません。しかし，テーブル排他なし TAM テーブルアクセス機能を使用する TAM テーブルではデッドロックとなることがあります。このため，テーブル排他あり TAM テーブルアクセス機能でレコードを更新，追加，または削除していた TAM テーブルを，テーブル排他なし TAM テーブルアクセス機能を使用するように変更する場合，UAP でアクセスするレコードの順序を統一してください。

テーブル排他あり TAM テーブルアクセス機能からテーブル排他なし TAM テーブルアクセス機能に変更してデッドロックとなる例を次の図に示します。

図 4-17　テーブル排他ありTAMテーブルアクセス機能からテーブル排他なしTAMテーブルアクセス機能に変更してデッドロックとなる例



UAP1 では，レコード 1，レコード 3 の順に削除し，UAP2 では，レコード 3，レコード 1 の順に更新するとします。テーブル排他あり TAM テーブルアクセス機能もテーブル排他なし TAM テーブルアクセス機能も図の(1)～(4)の順に実行されるとします。

テーブル排他あり TAM テーブルアクセス機能では，次の手順で動作します。

1. UAP1 のレコード 1 の削除で，更新目的のテーブル排他を確保します。
2. UAP2 のレコード 3 の更新では，手順 1 と排他の競合が発生し，参照目的のテーブル排他の確保で排他解除待ちとなります。
3. UAP1 のレコード 3 の削除では，排他は確保しません。
4. UAP1 のトランザクションが決着することによって手順 1 で確保した排他が解放されると，手順 2 の排他解除待ちが解けて UAP2 の処理が実行できます。

UAP1 のトランザクションが決着したあと，UAP2 の手順 2 では参照目的のテーブル排他と更新目的のレコード排他を確保し，手順 4 のレコード 1 の更新では，更新目的のレコード排他を確保します。

テーブル排他なし TAM テーブルアクセス機能では，次の手順で動作します。

1. UAP1 のレコード 1 の削除で，更新目的のレコード排他を確保します。
2. UAP2 のレコード 3 の更新で，更新目的のレコード排他を確保します。
3. UAP1 のレコード 3 の削除では，手順 2 と排他の競合が発生し，更新目的のレコード排他の確保で排他解除待ちとなります。
4. UAP2 のレコード 1 の更新では，手順 1 と排他の競合が発生し，更新目的のレコード排他の確保で排他解除待ちとなり，デッドロックが発生します。

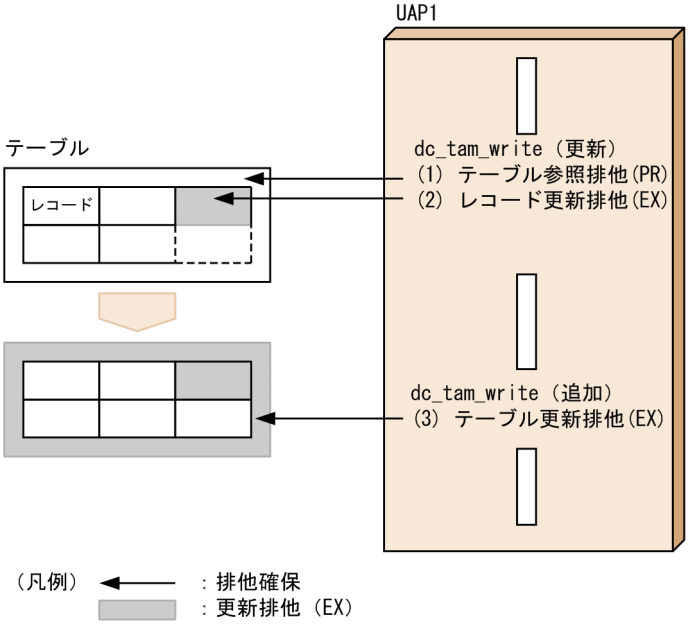
デッドロックを回避するには，UAP1 の手順 1 と手順 3 の順序を入れ替えるか，または，UAP2 の手順 2 と手順 4 の順序を入れ替えます。

(2) 排他確保の流れ

同一トランザクション内で同一テーブルに対する排他確保の様子を，レコードの「更新＋追加」という処理を例に説明します。

更新および追加の目的の排他確保の例を次の図に示します。

図 4-18 「更新＋追加」の例

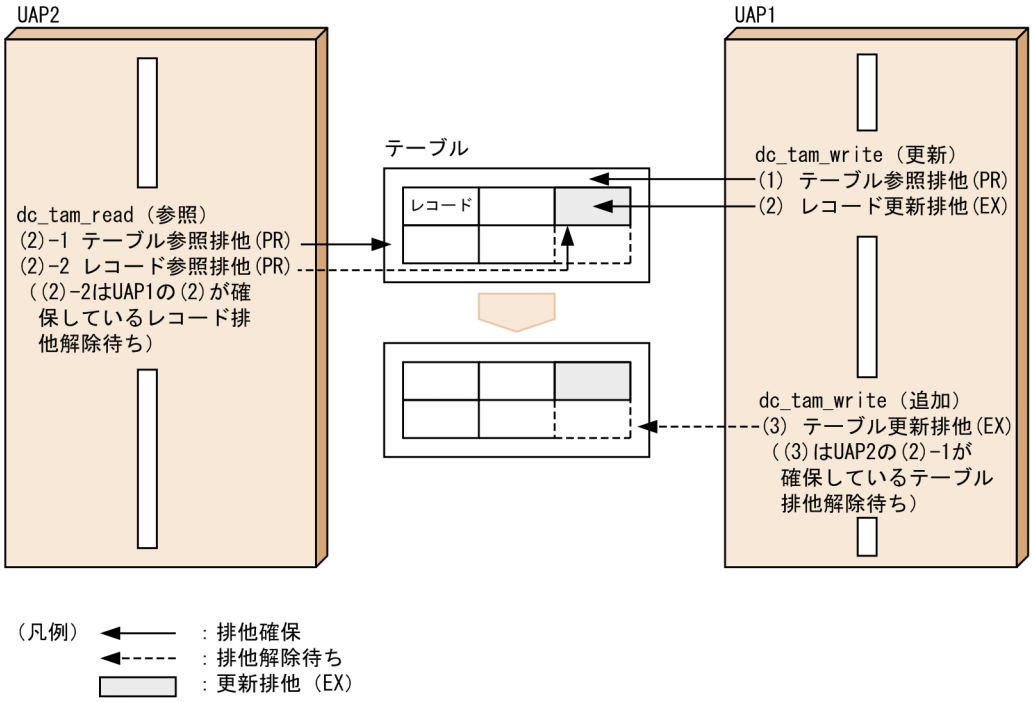


1. 更新目的の dc_tam_write で，この図の(1)，(2)に示す排他を確保してレコードを更新します。
   目的のレコードは，「追加削除できる更新型」TAM テーブルにあります。このトランザクションが終了するまで，テーブル内のレコード構成を別トランザクションに変更させないために，テーブル全体に参照排他（PR）を確保します。
2. 次に，追加目的の dc_tam_write で，この図の(3)に示す排他を確保してレコードを追加します。
   テーブル内の構成を変更するので，このトランザクションが終了するまでテーブルを別トランザクションに参照させないために，テーブル全体に更新排他（EX）を確保します。

3. 1.から 2.の一連の処理の中で，テーブルに対する排他を「参照排他（PR)」から「更新排他（EX)」に変更しようとすることになります。

   1.から 2.の処理の間で，デッドロックが発生する流れを次の図に示します。

図 4-19　デッドロック発生



   1.の処理のあとでかつ 2.の処理が行われる前であれば，別トランザクションは同一テーブルに対してこの図の(2)-1 に示すように参照排他（PR）を確保できます。このとき，別トランザクションが，本トランザクションで更新したレコードを排他確保しようとすると，テーブルに対して参照排他（PR）を確保したままレコードの排他解除待ちになります（この図の(2)-2)。

   その後，2.の処理が行われると，別トランザクションが同一テーブルに対して参照排他（PR）を確保しているために，テーブルに対して更新排他（EX）を確保できないで排他解除待ちになります。

4. 3.によって，二つのトランザクションがお互いの排他資源の解除待ちになり，デッドロックとなります。

   この図では，自トランザクション内の追加の処理で必要になるはずの資源（テーブル）の更新排他（EX）を確保しないで処理（2.の手前までの処理）を進め，別トランザクションにテーブルの排他確保を許してしまったためにデッドロックとなりました。あらかじめテーブルに対して更新排他（EX）を確保していれば，別トランザクションにテーブルの排他を確保させることはありません。

   このため，同一トランザクション内では同一テーブルに対して「追加（削除）およびその他 TAM アクセス」を行わないようにするなど，「(1)　トランザクションのデッドロック要因」を満たさないようにしてください。

(3)　デッドロックを避けるための注意

同一トランザクション内では同一テーブルに対して「追加（削除）およびその他 TAM アクセス」を行う必要があり，上記のようなデッドロックの危険がある場合は，該当するトランザクションでテーブルに対して更新排他（EX）を確保してから処理をするようにしてください。

テーブルに対して更新排他を確保するためには，「追加（削除）」を先に行うか，またはＣ言語であればテーブルに対してトランザクション内でテーブル排他でオープンするようにしてください。

4.3 IST サービス（TP1/Shared Table Access）

複数の OpenTP1 システムが，ノードをわたってテーブルを共用できる機能を **IST サービス**といいます。IST サービスで使えるテーブルを **IST テーブル**といいます。IST サービスを使うと，テーブルの実体がどのノードにあるかを意識しないで，UAP からデータを参照，更新できます。さらに，各ノードの業務状態を管理するために，**メール**として IST テーブルを使うこともできます。ただし，複数のノードにわたってデータを配布させる場合，次に示す条件の業務には IST サービスはお勧めできません。

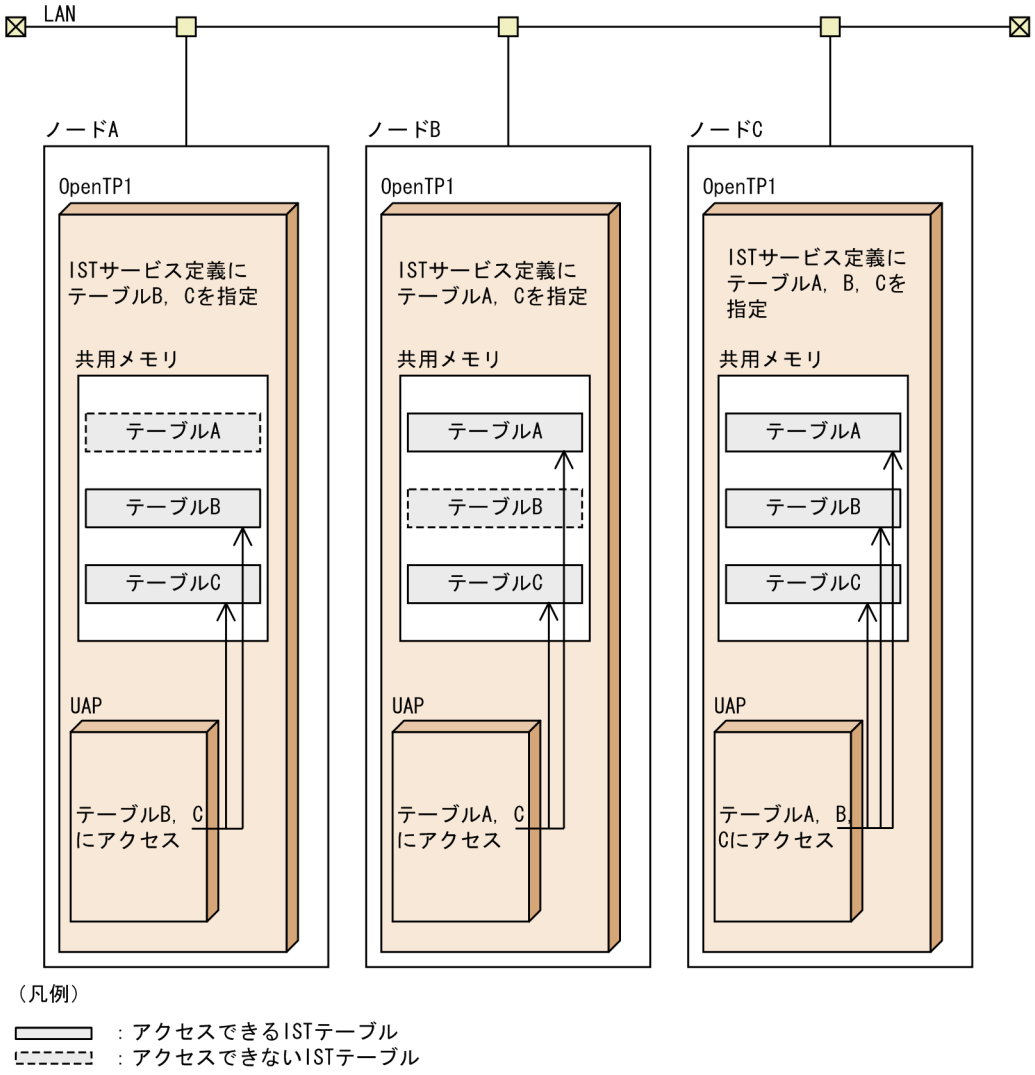
- データを即時に配布する必要がある業務
- 大量のデータを扱う必要がある業務
- 頻繁にデータを更新する業務

IST テーブルを使う場合，各ノードのシステムに TP1/Shared Table Access が組み込まれていることが前提となります。また，OpenTP1 の基本機能が TP1/Server Base の場合だけ，IST サービスを使えます。TP1/LiNK では IST サービスを使えません。

4.3.1 IST サービスのシステム構成

IST サービスを使用するには，ノード間の IST テーブル定義の指定を合わせてください。ノード間で IST テーブル定義の指定を合わせないと，KFCA25533-W メッセージが出力されます。ノード間で IST テーブル定義の指定が不一致の場合の例を次の図に示します。

図 4-20　ノード間で IST テーブル定義の指定が不一致の場合



この図の場合，ノード A，ノード B，およびノード C で IST テーブル定義に指定したテーブル名が合っていないため，予期しないテーブル情報を受信したとして，ノード A およびノード B で，OpenTP1 が終了するまで定期的に KFCA25533-W メッセージを出力し続けます。

4.3.2　IST テーブルの概要

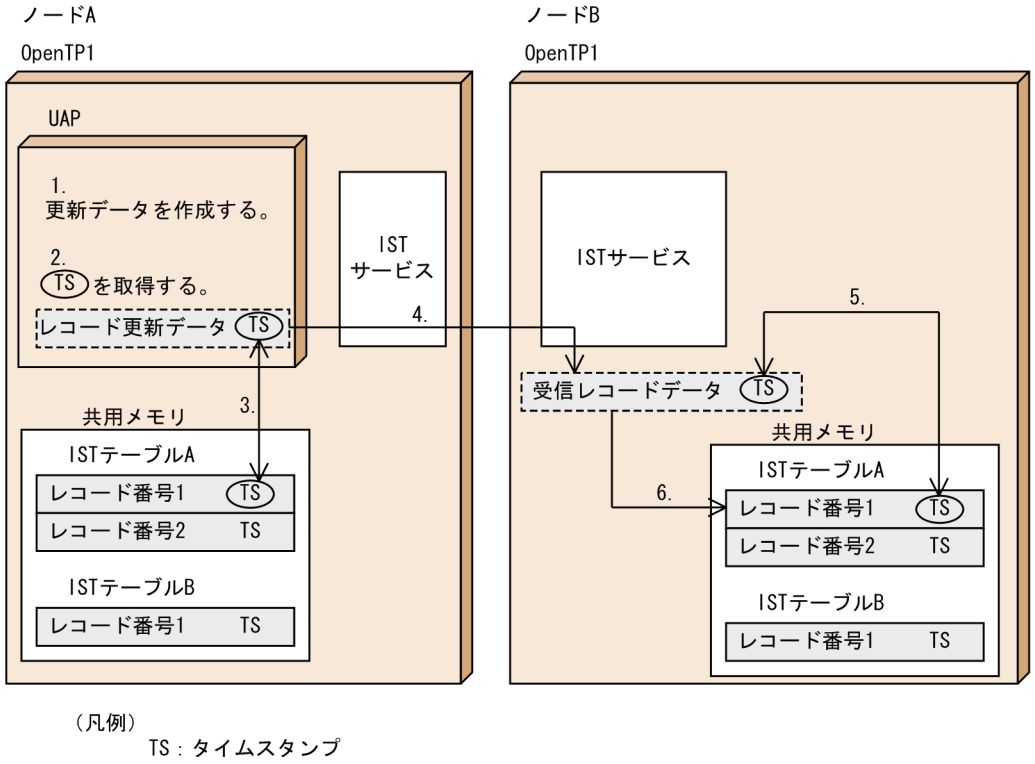
IST テーブルの概要について説明します。

(1)　IST テーブルへのアクセス環境

IST テーブルは，各ノードの共用メモリ上にあるテーブルです。テーブルの実体にあたるファイルはありません。そのため，UAP から IST テーブルへアクセスできるのは，**オンライン中**だけです。オフライン環境では IST テーブルにはアクセスできません。

また，複数のノードで IST サービスを使う場合には，各ノードの時刻を合わせておく必要があります。ノード間で時刻が一致していないと，あるノードで更新したデータに対して，別のノードからの更新が反映されないことがあります。IST サービスで複数のノードの IST レコード（IST テーブル中のレコード）を更新する処理の流れを次の図に示します。

図 4-21　IST レコードの更新処理



（凡例）
TS：タイムスタンプ

1. ノード A の IST テーブル A の IST レコード（レコード番号 1）を更新するレコード更新データを作成します。
2. 現在の時刻（マシン時刻：マイクロ秒精度）を取得し，レコード更新データにタイムスタンプとして付与します。
3. ノード A の共用メモリ上の該当する IST レコードに設定されているタイムスタンプとレコード更新データに付与したタイムスタンプとを比較します。

   レコード更新データの方が新しい場合は，共用メモリ上の IST レコードを更新します。レコード更新データの方が古い場合は，共用メモリ上の IST レコードを更新しません。なお，IST レコードを更新しない場合も，dc_ist_write 関数は正常にリターンします。
4. 共用メモリ上の IST レコードを更新した場合，ノード A で IST レコードを更新したことをノード B の IST サービスへ通知します。このとき，IST レコードと IST レコードに付与したタイムスタンプも通知します。
5. 更新された IST レコードを受信したノード B の IST サービスは，ノード内の該当する IST レコードに設定されているタイムスタンプと受信した IST レコードのタイムスタンプとを比較します。
6. 5.の結果，受信した IST レコードのタイムスタンプの方が新しいと判断した場合だけ，ノード B の該当する IST レコードを，受信した IST レコードの情報に更新します。

上記のように，IST サービスでは，IST レコードを更新するか，またはそのままとするかを，タイムスタンプを基に判断しています。次のような場合は，最新の更新データが IST レコードに反映されないことがあります。

- ノード A のマシン時刻がノード B のマシン時刻よりも進んでいる場合

  ノード A で IST レコードを更新したあとに，ノード B から同一の IST レコードの更新を通知されても，ノード A の IST レコードに設定されたタイムスタンプの方が新しいと判断します。そのため，ノード B で更新された IST レコードの情報がノード A の IST レコードに反映されません。

  また，ノード A で更新した IST レコードをノード B へ通知したときに，通知した IST レコードのタイムスタンプの方が新しいと判断されるため，ノード B の該当する IST レコードは，実際には最新の情報であっても，通知した IST レコードの情報に更新されてしまいます。

- ノード A のマシン時刻がノード B のマシン時刻よりも遅れている場合

  - ノード B が IST レコードを更新して，その IST レコードの情報がすでにノード A に通知されているとき

    ノード B で IST レコードを更新したあとに，ノード A で同一の IST レコードを更新しても，更新処理をしないで dc_ist_write 関数が正常リターンします。

  - ノード B が IST レコードを更新したが，その IST レコードの情報がまだノード A に通知されていないとき

    ノード B で IST レコードを更新したあとに，ノード A で同一の IST レコードを更新すると，ノード A の更新情報で，いったん IST レコードを更新しますが，そのあとにノード B から通知された IST レコードのタイムスタンプの方が新しいと判断するため，ノード B から通知された IST レコードの情報を，ノード A の IST レコードに反映してしまいます。

(2) IST テーブルの構造

UAP から IST テーブルを参照，更新するときは，**レコード単位**でアクセスします。IST テーブルは，複数のレコードから構成されます。UAP の処理では，一つのレコードへアクセスすることも，複数のレコードへ一括してアクセスすることもできます。

4.3.3 IST テーブルへのアクセス手順

UAP から IST テーブルへアクセスするときの手順について説明します。なお，IST テーブルへのアクセスは，トランザクションの関数でコミット，ロールバックできません。

(1) IST テーブルのオープン

UAP から IST テーブルにアクセスする場合は，まず IST テーブルをオープンします。IST テーブルをオープンするときは，**dc_ist_open 関数【CBLDCIST('OPEN')】**を呼び出します。IST テーブルをオープンすると，**テーブル記述子**がリターンされます。IST テーブルのオープン以降の処理では，テーブル記述子を関数に設定してアクセスします。テーブル記述子は，オープン以降の処理でも UAP で保持しておいてください。

(2)　レコードの参照／更新手順

IST テーブルのレコードを入力するときは，dc_ist_read 関数【CBLDCIST('READ')】を呼び出します。IST テーブルのレコードへデータを出力するときは，dc_ist_write 関数【CBLDCIST('WRIT')】を呼び出します。dc_ist_read 関数，dc_ist_write 関数を呼び出すときは，dc_ist_open 関数でリターンされた**テーブル記述子**を引数に設定します。

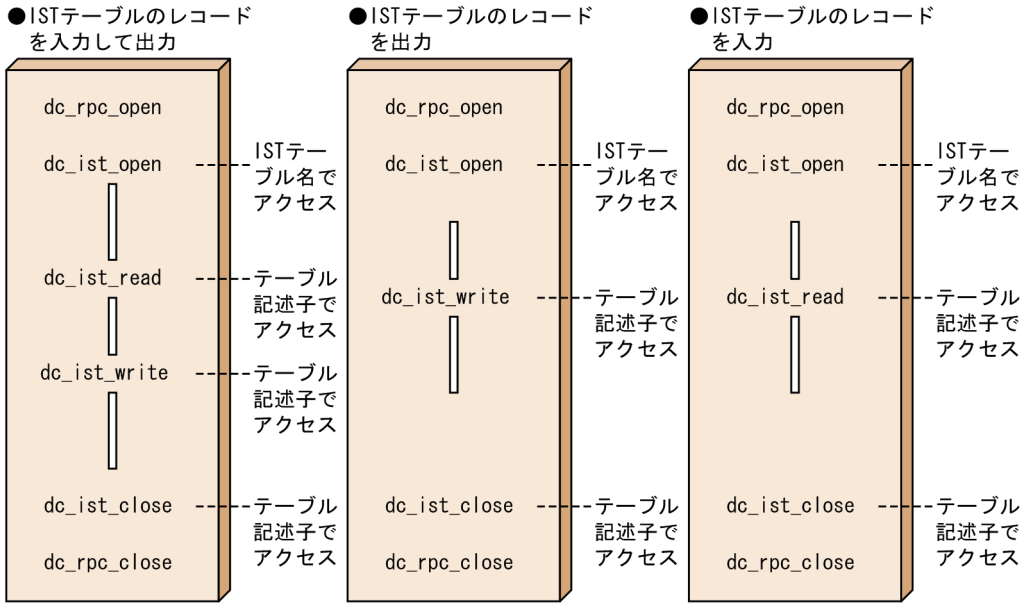
レコードを入力または出力するときは，複数のレコードのキー値を一括して指定できます。キー値は，構造体として関数に設定します。この構造体は，複数個指定できます。

(3)　IST テーブルのクローズ

IST テーブルをクローズするときは，dc_ist_close 関数【CBLDCIST('CLOS')】を呼び出します。
dc_ist_close 関数を呼び出すときは，dc_ist_open 関数でリターンしたテーブル記述子を引数に設定します。

IST テーブルへのアクセス手順を次の図に示します。

図 4-22　IST テーブルへのアクセス手順



4.3.4　IST テーブルの排他制御

IST テーブルでは，UAP から呼び出した関数ごとに排他制御しています。データの入力から更新まで IST テーブルを占有する制御はしていません。そのため，一つのテーブルに複数の UAP からアクセスした場合でも，デッドロックが起こることはありません。

4.4 ISAM ファイルサービス（ISAM，ISAM/B）

索引順編成ファイルを管理する，ISAM ファイルサービスについて説明します。機能については，マニュアル「索引順編成ファイル管理 ISAM」を参照してください。

4.4.1 ISAM ファイルの概要

索引順編成ファイルは，キーを管理する**インデクス部**と，データを格納する**データファイル部**から構成されます。キーを使用して，**順処理（シーケンシャルアクセス）**や**乱処理（ランダムアクセス）**ができます。

ISAM ファイルの操作には，**ライブラリ関数を UAP から呼び出す方法**と，**ユティリティコマンドを実行して管理する方法**があります。

4.4.2 ISAM サービスの種類

OpenTP1 の UAP では，次に示す ISAM ファイルサービスを使えます。

- ISAM
- ISAM/B

ISAM は，TP1/Server Base と TP1/LiNK で使えます。ISAM/B は，TP1/Server Base の場合だけ使えます。OpenTP1 の基本機能が TP1/LiNK の場合は，ISAM/B は使えません。

(1) ISAM

ISAM ファイルを，通常のファイルとして使います。OpenTP1 のトランザクション処理とは同期しません。

(2) ISAM/B

ISAM ファイルをトランザクション処理と同期して使う機能です。ISAM/B を使うと，トランザクション処理のコミット，またはロールバックで，ファイルの整合性が保てるようになります。

(a) ISAM/B の前提となる製品

ISAM ファイルを ISAM/B として使う場合は，ISAM に加えて，ISAM トランザクション機能（**ISAM/B**）が前提となります。

(b) ファイルを作成する領域

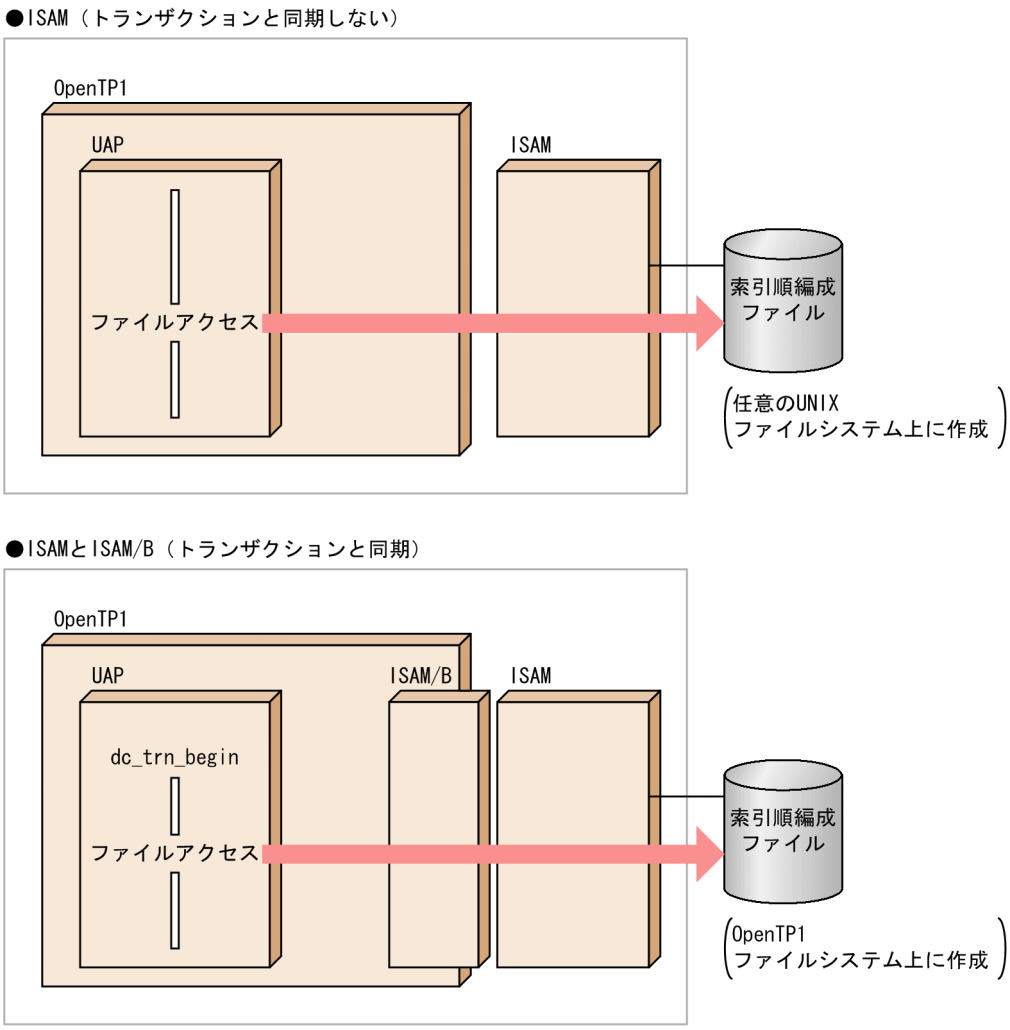
ISAM/B で使う ISAM ファイルは，OpenTP1 ファイルシステムとして割り当てた領域に作成します。

(c)　OpenTP1 のファイルサービス（TP1/FS/xxx）との違い

ISAM/B では，ロックサービスを使いません。そのため，デッドロックが起こっても，OpenTP1 のロックサービス機能（優先順位による縮退やデッドロック情報の出力）は使えません。

ISAM ファイルサービスの形態を次の図に示します。

図 4-23　ISAM ファイルサービスの形態



4.5　データベースにアクセスする場合

OpenTP1 の UAP で使うユーザファイルに，データベースマネジメントシステム（DBMS）を適用した場合について説明します。

4.5.1　OpenTP1 のトランザクション処理との関係

DBMS を使う場合，X/Open で規定した DTP モデルの XA インタフェースをサポートした DBMS かどうかで，OpenTP1 のトランザクションと連携できるかどうか決まります。

(1)　XA インタフェースをサポートした DBMS の場合

XA インタフェースをサポートした DBMS の場合は，OpenTP1 のトランザクションと同期を取って更新できます。同期を取る場合は，OpenTP1 の同期点を制御する関数（dc_trn_begin 関数，dc_trn_unchained_commit 関数，tx_begin()，tx_commit()　など）を使います。DBMS で提供する同期点を制御する機能は，OpenTP1 の同期点を制御する関数と併用できません。DBMS の同期点を制御する機能を使った場合，リソースの不整合が起こってしまう場合があります。

OpenTP1 のトランザクション処理で制御できる DBMS は，XA インタフェースをサポートした製品に限ります。

XA インタフェースをサポートしている場合，複数のデータベースへアクセスする UAP では，複数のデータベースを，整合性を保ちながら更新できます。次に示す OpenTP1 のリソースマネジャは，XA インタフェースをサポートしています。

- TP1/FS/Direct Access（DAM ファイルサービス）
- TP1/FS/Table Access（TAM ファイルサービス）
- ISAM，ISAM/B（ISAM ファイルサービス）
- TP1/Message Control（メッセージ送受信機能（MCF））
- TP1/Message Queue（メッセージキューイング機能）

そのため，XA インタフェースに準拠した DBMS と，OpenTP1 のリソースマネジャの両方にアクセスする UAP でも，OpenTP1 のトランザクションとして処理できます。障害が原因で UAP が異常終了した場合や，OpenTP1 を再開始（リラン）した場合でも，DBMS と OpenTP1 リソースマネジャの両方のトランザクションを，OpenTP1 で決着します。

(2)　XA インタフェースをサポートしていない，または XA インタフェースで OpenTP1 と連携していない DBMS の場合

XA インタフェースをサポートしていない DBMS の場合，DBMS へのアクセスはできますが，OpenTP1 のトランザクションとは同期を取れません。

XA インタフェースで OpenTP1 と連携していないため，DBMS へのアクセス中に，障害が原因で UAP が異常終了した場合や，OpenTP1 を再開始（リラン）した場合には，OpenTP1 から DBMS へトランザクションの決着を指示しません。そのため，DBMS 独自の機能でトランザクションを回復する必要があります。

4.5.2　XA インタフェースでデータベースにアクセスする場合の準備

XA インタフェースをサポートした DBMS を，OpenTP1 と XA インタフェースで連携して使う場合に準備する項目を次に示します。この準備は，OpenTP1 提供以外のリソースマネジャを使う場合に必要です。

(1)　OpenTP1 への登録

OpenTP1 提供以外のリソースマネジャの各種名称を登録します。OpenTP1 へは，次に示すどちらかの方法で登録します。

- dcsetup コマンドで OpenTP1 をセットアップ後，**trnlnkrm コマンド**を実行する。
- **拡張 RM 登録定義**を作成する。

拡張 RM 登録定義を作成しておくと，dcsetup コマンドで OpenTP1 をセットアップしたあとに trnlnkrm コマンドを実行しなくてもよくなります。trnlnkrm コマンドの使い方については，マニュアル「OpenTP1 運用と操作」を参照してください。拡張 RM 登録定義の指定方法については，マニュアル「OpenTP1 システム定義」を参照してください。

(2)　UAP のリンケージ

UAP の実行形式ファイルを作成するときに，トランザクション制御用オブジェクトファイル，および DBMS のライブラリとオブジェクトモジュールをリンケージする必要があります。

トランザクション制御用オブジェクトファイルは，**trnmkobj コマンド**を実行して作成します。trnmkobj コマンドの使い方については，マニュアル「OpenTP1 運用と操作」を参照してください。

(3)　システム定義

DBMS を使う場合，トランザクションサービス定義に **trnstring 形式の定義**を，必要に応じてユーザサービス定義，およびユーザサービスデフォルト定義に **trnrmid 形式の定義**をする必要があります。指定する内容には，DBMS 固有の項目もあります。このような項目は，使用する DBMS のマニュアルを参照してください。

trnstring，trnrmid 形式の定義を指定すると，一つのリソースマネジャを複数の制御単位に分け，接続するユーザ名称などを変更してリソースマネジャに接続することもできます（リソースマネジャ接続先選択機能）。リソースマネジャ接続先選択機能については，「4.5.3　リソースマネジャ接続先選択機能」を参照してください。

trnstring，trnrmid 形式の定義については，マニュアル「OpenTP1 システム定義」を参照してください。

また，OpenTP1 以外の RM を使う場合，トランザクションサービス定義に，**set 形式の定義**をして，スレッドスタック領域のサイズを拡張する必要があります。

set 形式の定義については，マニュアル「OpenTP1 システム定義」を参照してください。

(4) 環境変数

DBMS を使う場合，特定の環境変数が必要になる場合があります。その場合，トランザクションサービス定義，ユーザサービス定義，またはユーザサービスデフォルト定義に，**putenv 形式**の定義をする必要があります。

putenv 形式の定義については，マニュアル「OpenTP1 システム定義」を参照してください。

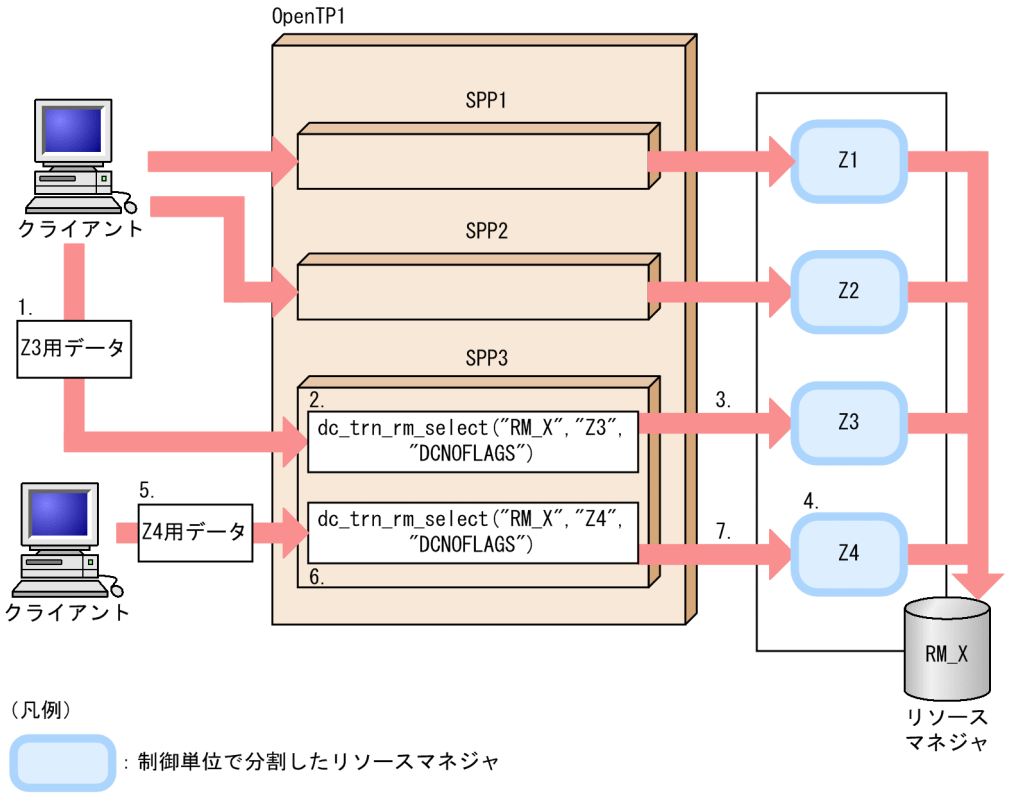
4.5.3 リソースマネジャ接続先選択機能

一つのリソースマネジャを複数の制御単位に分け，接続するユーザ名称などを変更してリソースマネジャに接続できます。この機能を，**リソースマネジャ接続先選択機能**といいます。リソースマネジャ接続先選択機能を使用すると，RPC メッセージの情報に応じてリソースマネジャの接続先を動的に変更するように SPP を一度実装すれば，その後の業務拡大に伴ってデータベースのテーブルやサーバを追加することになっても，SPP を修正しないでデータベースの接続先を変更できます。これによって，肥大化するテスト工数や運用コストを削減できます。

この機能は，SPP，および SUP でだけ使用できます。また，OpenTP1 提供以外のリソースで使用できます。

リソースマネジャ接続先選択機能使用時にデータベースの接続先を追加する場合の例を示します。

図 4-24　リソースマネジャ接続先選択機能使用時にデータベースの接続先を追加する場合の例



1. クライアントが SPP3 にサービス要求を送信します。
2. dc_trn_rm_select 関数をリソースマネジャ RM_X の Z3 で発行し，リソースマネジャの接続先を設定します。
3. Z3 に対して接続，更新処理を実施します。
4. 業務量の拡大により，Z4 を追加します。
5. SPP3 に，Z4 用のデータで新たなサービス要求を送信します。
6. dc_trn_rm_select 関数をリソースマネジャ RM_X の Z4 で発行し，リソースマネジャの接続先を設定します。
7. Z4 に対して接続，更新処理を実施します。

この図では SPP1 と SPP2 がそれぞれ Z1 と Z2 にだけ接続していますが，SPP3 は dc_trn_rm_select 関数【CBLDCTRN('RMSELECT')】を発行することで接続先を変更できるようにしています。

この機能を使用するには，トランザクションサービス定義の trnstring 定義コマンド，ユーザサービス定義およびユーザサービスデフォルト定義の trnrmid 定義コマンドを指定する必要があります。各定義コマンドについては，マニュアル「OpenTP1 システム定義」を参照してください。

(1)　dc_trn_rm_select 関数の動作

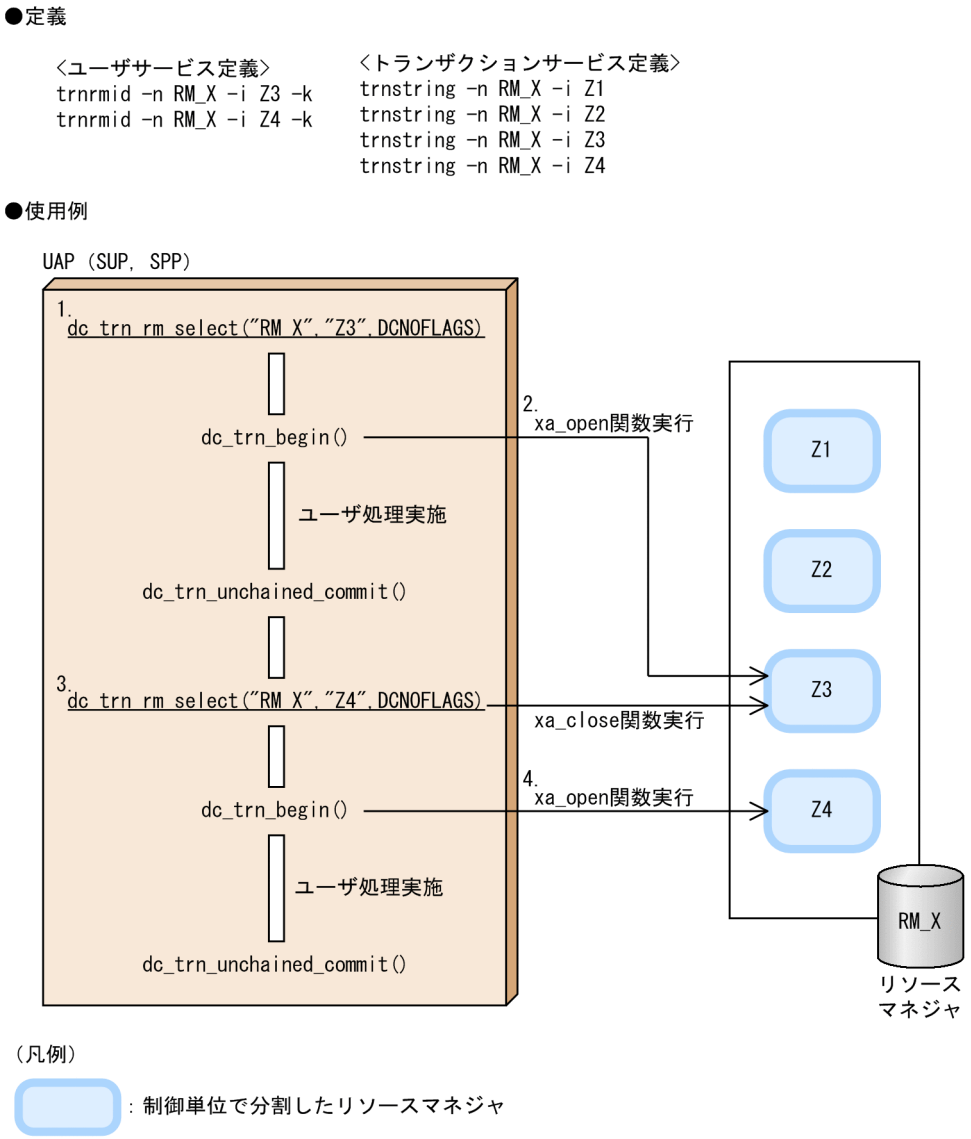
リソースマネジャの接続先を UAP の処理内でプロセス，またはトランザクションごとに選択する場合は，dc_trn_rm_select 関数【CBLDCTRN('RMSELECT')】を使用します。

dc_trn_rm_select 関数についてはマニュアル「OpenTP1 プログラム作成リファレンス C 言語編」を，CBLDCTRN('RMSELECT')についてはマニュアル「OpenTP1 プログラム作成リファレンス COBOL 言語編」を参照してください。

(a)　dc_trn_rm_select 関数の使用例

dc_trn_rm_select 関数の使用例を次の図に示します。

図 4-25　dc_trn_rm_select 関数の使用例



1. dc_trn_rm_select 関数を使用して，接続先となるリソースマネジャ名とリソースマネジャ拡張子を設定します。
2. dc_trn_rm_select 関数の処理の中で，接続先として指定されたリソースマネジャをオープンします。
3. リソースマネジャの接続先を変更するため，dc_trn_rm_select 関数を再度実行します。説明 1.で指定されたリソースマネジャと接続先が異なる場合は，説明 1.でオープンしたリソースマネジャをクローズします。

4. dc_trn_rm_select 関数の処理の中で，接続先として指定されたリソースマネジャをオープンします。

(b) dc_trn_rm_select 関数の発行有無による動作の違い

trnrmid 定義コマンドに-k オプションを指定したリソースマネジャは，dc_trn_rm_select 関数で設定するまで，トランザクションから該当するリソースマネジャに接続できません。

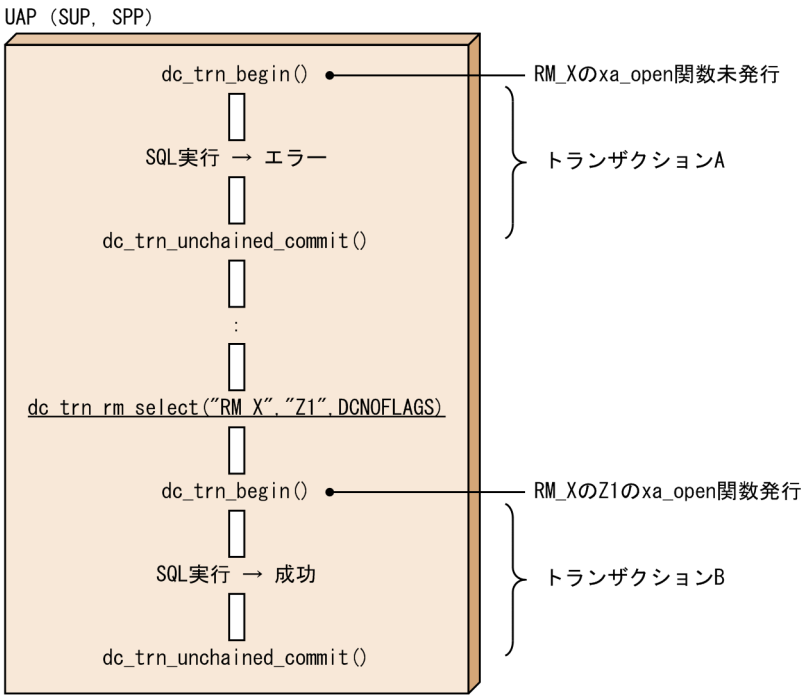
dc_trn_rm_select 関数の発行有無による動作の違いについて，次に示します。

図 4-26 dc_trn_rm_select 関数の発行有無による動作の違い

●定義

```
〈ユーザサービス定義〉          〈トランザクションサービス定義〉
trnrmid -n RM_X -i Z1 -k      trnstring -n RM_X -i Z1, Z2
trnrmid -n RM_X -i Z2 -k
```

●動作



最初の dc_trn_begin 関数実行時は，dc_trn_rm_select 関数を発行していない状態であるため，リソースマネジャ RM_X の Z1 または Z2 に対する xa_open 関数は発行されません。しかし，2 回目の dc_trn_begin 関数実行時には，dc_trn_rm_select 関数で接続先が設定されたため，リソースマネジャ RM_X の Z1 に対する xa_open 関数が発行されます。

トランザクション A はリソースマネジャ RM_X に接続できませんが，トランザクション B はリソースマネジャ RM_X の Z1 に接続できます。

(c) dc_trn_rm_select 関数がエラーリターンした場合の動作

dc_trn_rm_select 関数がエラーになった場合は，指定したリソースマネジャは接続先として設定されません。

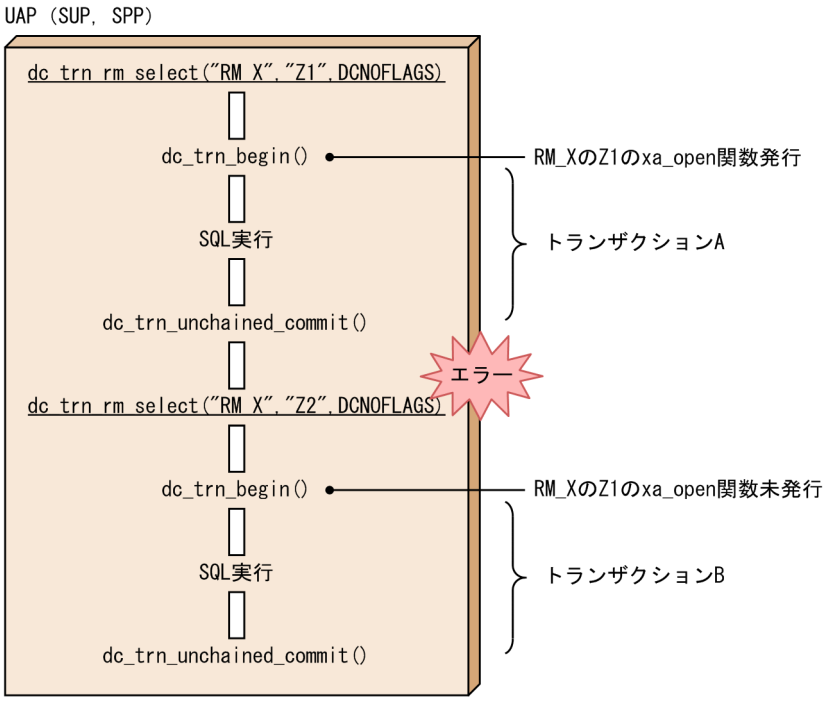
dc_trn_rm_select 関数がエラーになった場合の動作について，次に示します。

図 4-27　dc_trn_rm_select 関数がエラーになった場合の動作

●定義

```
〈ユーザサービス定義〉          〈トランザクションサービス定義〉
trnrmid -n RM_X -i Z1 -k      trnstring -n RM_X -i Z1, Z2
trnrmid -n RM_X -i Z2 -k
```

●動作



2 回目の dc_trn_rm_select 関数がエラーリターンしたため，リソースマネジャ RM_X の Z2 は接続先として設定されないで，先に設定されたリソースマネジャ RM_X の Z1 がそのまま接続先として継続されます。

トラザクション A と B は，リソースマネジャ RM_X の Z1 に接続できます。

(2)　trn_rm_open_close_scope オペランド指定時の動作

リソースマネジャの xa_open 関数と xa_close 関数を発行するタイミングは，トランザクションサービス定義の trn_rm_open_close_scope オペランドで変更できます。ただし，リソースマネジャ接続先選択機能を使用した場合，xa_open 関数の発行タイミングが trn_rm_open_close_scope オペランドの指定と異なることがあります。

trn_rm_open_close_scope オペランドの指定値別に，動作について説明します。

(a)　trn_rm_open_close_scope オペランドに process を指定した場合

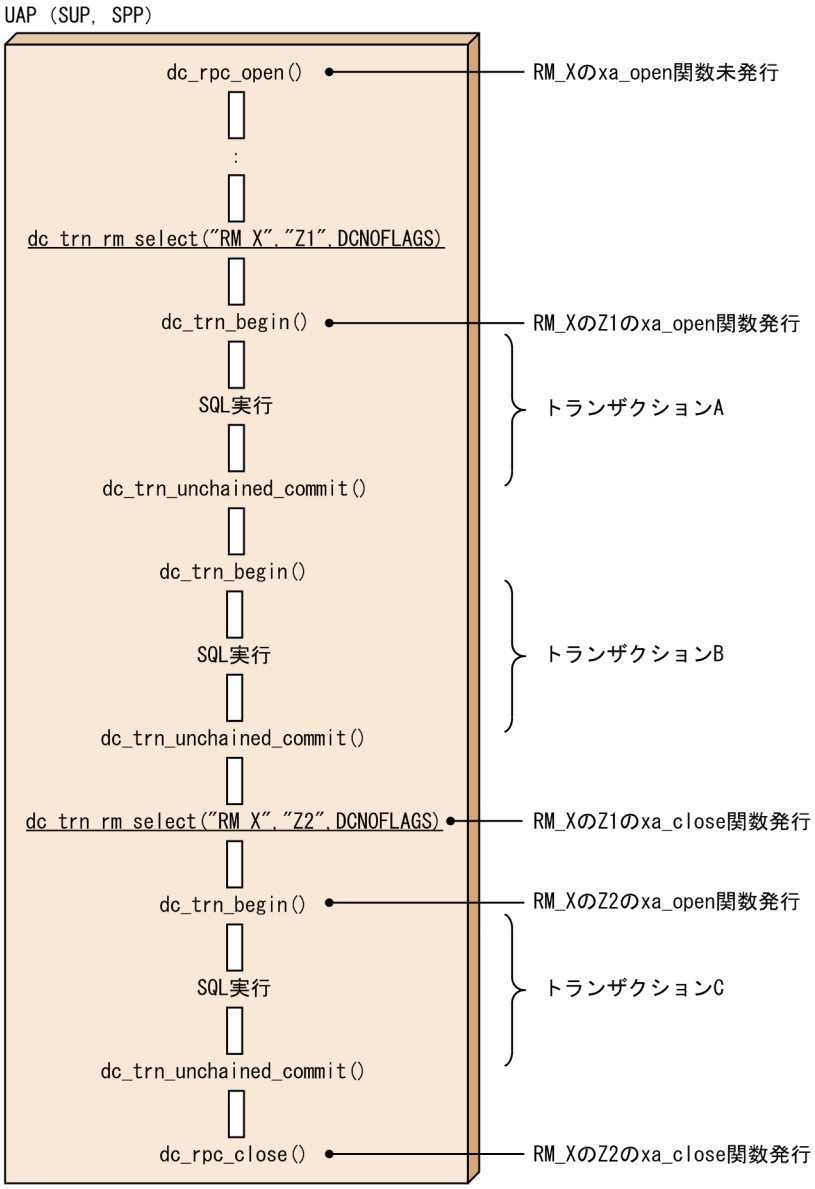
trn_rm_open_close_scope オペランドに process を指定した場合の動作を，次に示します。

図 4-28　trn_rm_open_close_scope オペランドに process を指定した場合の動作

●定義

```
〈ユーザサービス定義〉          〈トランザクションサービス定義〉
trn_rm_open_close_scope      trnstring -n RM_X -i Z1,Z2
          =process
trnrmid -n RM_X -i Z1 -k
trnrmid -n RM_X -i Z2 -k
```

●動作



trnrmid 定義コマンドに-k オプションが指定されたリソースマネジャでは，dc_rpc_open 関数では xa_open 関数が発行されません。dc_trn_rm_select 関数で設定されたリソースマネジャ RM_X の Z1 に対して，最初の dc_trn_begin 関数で xa_open 関数が発行されます。2 回目の dc_trn_rm_select 関数でリソースマネジャ RM_X の Z2 が設定されたため，そのタイミングでいったんリソースマネジャ RM_X の Z1 に対して xa_close 関数を発行します。その後，dc_trn_begin 関数で RM_X の Z2 に対して xa_open 関数が発行されます。

トランザクション A と B はリソースマネジャ RM_X の Z1 に接続可能で，トランザクション C はリソースマネジャ RM_X の Z2 に接続可能です。

(b)　trn_rm_open_close_scope オペランドに transaction を指定した場合

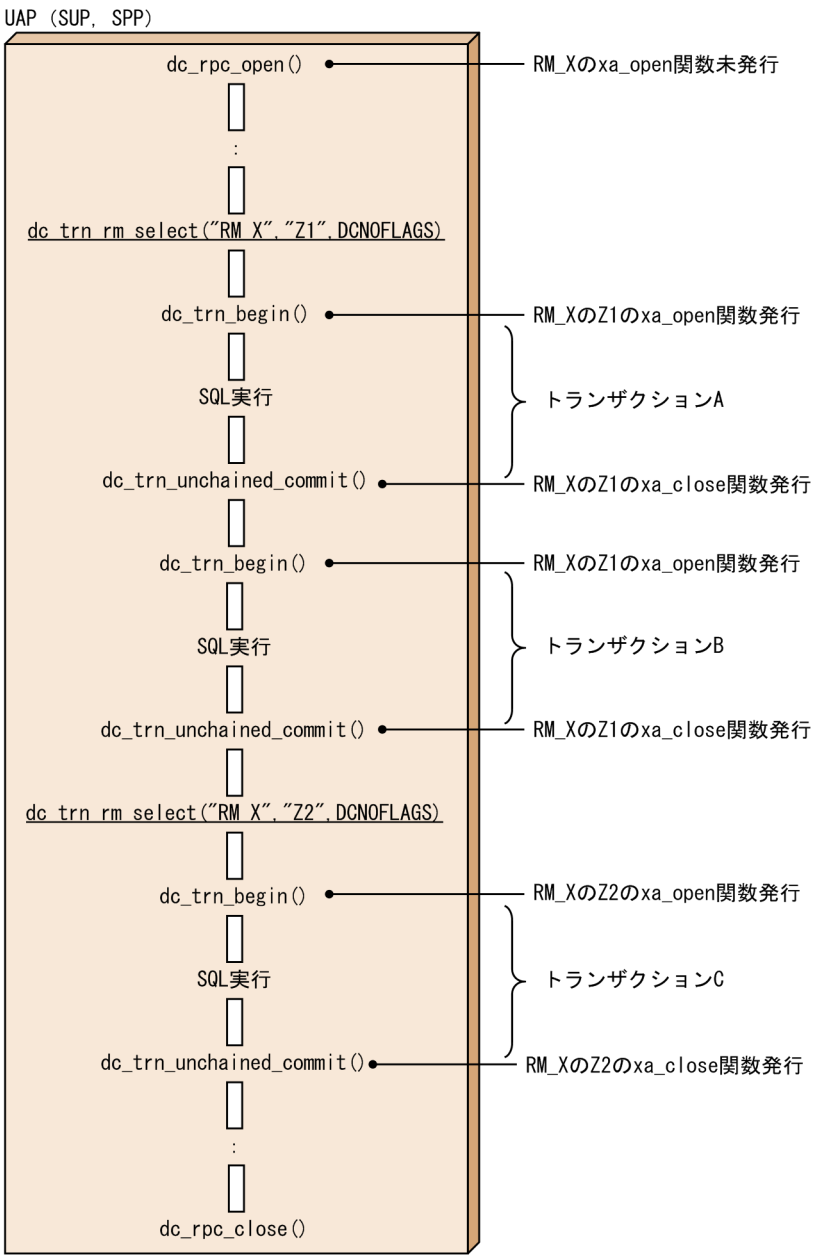
trn_rm_open_close_scope オペランドに transaction を指定した場合の動作を，次に示します。

図 4-29　trn_rm_open_close_scope オペランドに transaction を指定した場合の動作

●定義

```
〈ユーザサービス定義〉          〈トランザクションサービス定義〉
trn_rm_open_close_scope       trnstring -n RM_X -i Z1,Z2
            =transaction
trnrmid -n RM_X -i Z1 -k
trnrmid -n RM_X -i Z2 -k
```

●動作



trnrmid 定義コマンドに-k オプションが指定されたリソースマネジャでは，dc_rpc_open 関数では xa_open 関数が発行されません。dc_trn_rm_select 関数で設定されたリソースマネジャ RM_X の Z1 に対して，最初の dc_trn_begin 関数で xa_open 関数が発行され，dc_trn_unchained_commit 関数で xa_close 関数が発行されます。その後，トランザクションは接続先が変更されるまで同じリソースマネジャ RM_X の Z1 に接続します。2 回目の dc_trn_rm_select 関数でリソースマネジャ RM_X の Z2 が設定されますが，RM_X の Z1 に対する xa_close 関数は前回トランザクション完了時に発行済みであるため，このタイミングでリソースマネジャ RM_X の Z1 に対する xa_close 関数は発行されません。

トランザクション A と B はリソースマネジャ RM_X の Z1 に接続可能で，トランザクション C は RM_X の Z2 に接続可能です。

(3) 複数種類のリソースマネジャを指定する場合の動作

一つのトランザクションブランチから同一リソースマネジャへの接続は一つだけとなりますが，リソースマネジャが異なる場合は複数接続することができます。

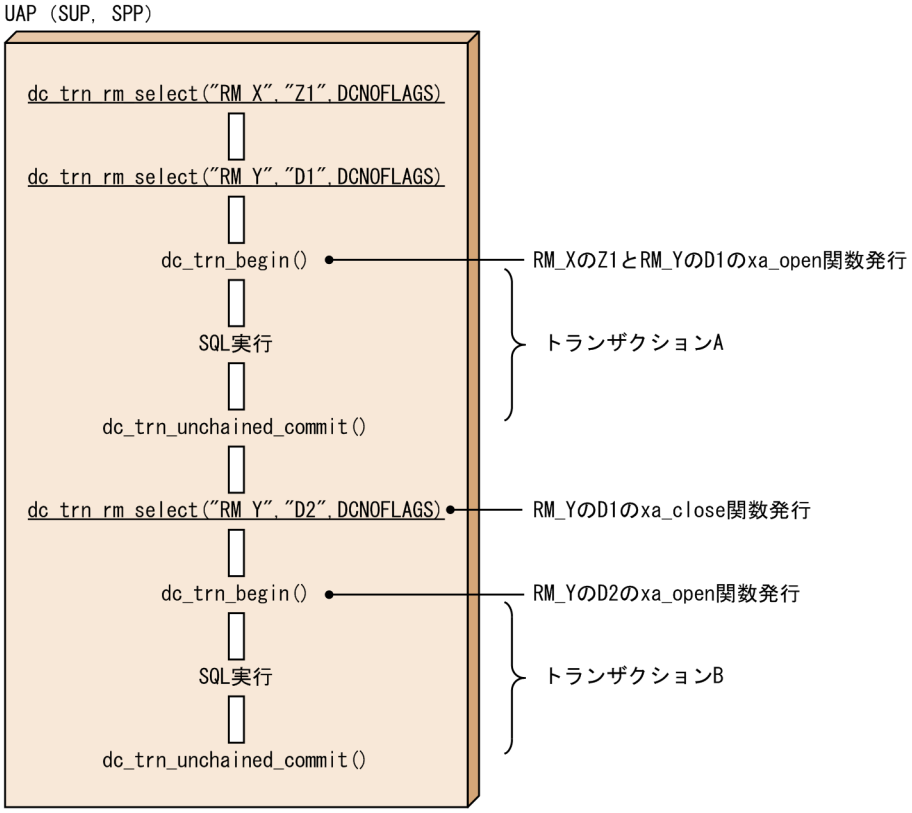
複数種類のリソースマネジャを指定する場合の動作を，次に示します。

図 4-30　複数種類のリソースマネジャを指定する場合の動作

●定義

```
〈ユーザサービス定義〉          〈トランザクションサービス定義〉
trn_rm_open_close_scope      trnstring -n RM_X -i Z1,Z2
            =process         trnstring -n RM_Y -i D1,D2
trnrmid -n RM_X -i Z1 -k
trnrmid -n RM_X -i Z2 -k
trnrmid -n RM_Y -i D1 -k
trnrmid -n RM_Y -i D2 -k
```

●動作



dc_trn_rm_select 関数でリソースマネジャ RM_X の Z1 と，リソースマネジャ RM_Y の D1 が設定されたため，最初の dc_trn_begin 関数でそれぞれのリソースマネジャの xa_open 関数を発行します。3 回目の dc_trn_rm_select 関数は，リソースマネジャ RM_Y の D2 が指定されたためリソースマネジャ RM_Y の D1 に対する xa_close 関数を発行しますが，種別が異なるリソースマネジャである RM_X の Z1 に対する xa_close 関数は発行しません。

トランザクション A はリソースマネジャ RM_X の Z1 とリソースマネジャ RM_Y の D1 に接続可能で，トランザクション B はリソースマネジャ RM_X の Z1 とリソースマネジャ RM_Y の D2 に接続可能です。

(4)　trnrmid 定義コマンドの-k オプションの指定有無による動作の違い

trnrmid 定義コマンドに-k オプションを指定しない場合，trn_rm_open_close_scope オペランドの指定によってプロセス開始時，またはトランザクション開始時に xa_open 関数が発行されます。そのため，同一リソースマネジャについて-k オプションがあるものとないものを指定すると，-k オプション指定のリソースマネジャを dc_trn_rm_select 関数で接続先として設定しても，トランザクション開始時に xa_open 関数がエラーとなります。

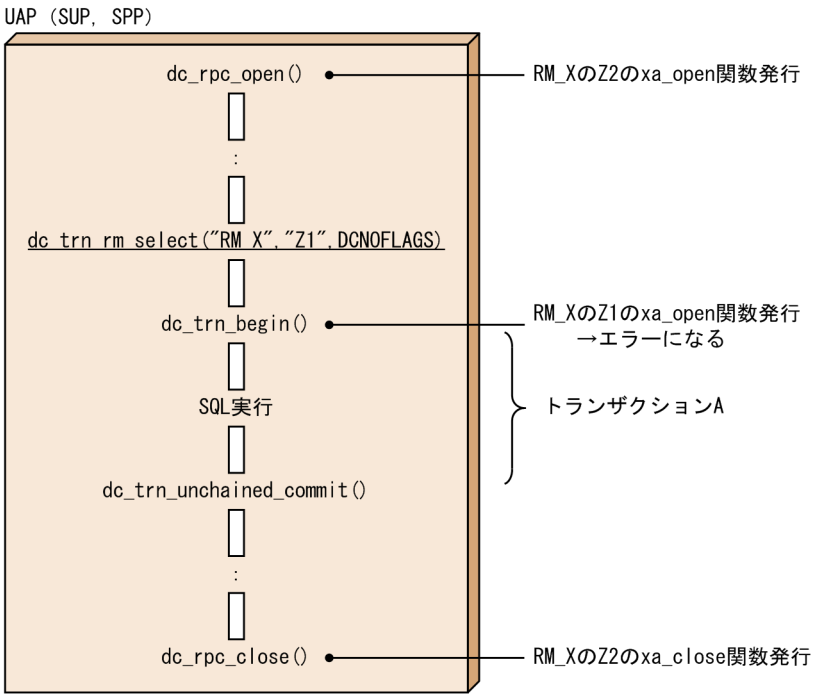
trnrmid 定義コマンドの-k オプションの指定有無による動作の違いについて，次の図に示します。

図 4-31　trnrmid 定義コマンドの-k オプションの指定有無による動作の違い

●定義

```
〈ユーザサービス定義〉          〈トランザクションサービス定義〉
trnrmid -n RM_X -i Z1 -k      trnstring -n RM_X -i Z1, Z2
trnrmid -n RM_X -i Z2
```

●動作



trnrmid 定義コマンドに-k オプションが指定されていないため，リソースマネジャ RM_X の Z2 に対する xa_open 関数は通常どおり dc_rpc_open 関数で発行します。dc_trn_rm_select 関数でリソースマネジャ RM_X の Z1 が設定されたため，dc_trn_begin 関数で xa_open 関数を発行しますが，リソースマネジャ RM_X の Z2 に対して xa_open 関数を発行済みであるため，この xa_open 関数はエラーとなります。

トランザクション A は，リソースマネジャ RM_X の Z2 に接続できます。

リソースマネジャ接続先選択機能を使用する場合は，ユーザサービス定義に指定する同一リソースマネジャの，すべての trnrmid 定義コマンドで-k オプションの指定を同じにしてください。

4.6　資源の排他制御

ユーザの任意の資源を，OpenTP1 の UAP から排他制御する方法について説明します。任意の資源を確保する場合，OpenTP1 の UAP から dc_lck_get 関数【CBLDCLCK('GET ')】を呼び出します。

任意の資源の排他制御は，OpenTP1 の基本機能が TP1/Server Base の場合だけ使えます。TP1/LiNK では，任意の資源の排他制御は使えません。

資源の排他制御は，トランザクションとして実行している処理で使います。それぞれのトランザクションから排他を指定することで，資源は正しく更新されて，UAP のトランザクション処理同士で排他できるようになります。

DAM ファイルの排他制御については「4.1　DAM ファイルサービス（TP1/FS/Direct Access)」を，TAM ファイルの排他制御については「4.2　TAM ファイルサービス（TP1/FS/Table Access)」を参照してください。

4.6.1　排他の対象となる資源

排他の対象になるのは，オペレーティングシステム内で固有の名称を定義してある，ファイルなどの資源です。ユーザ固有の資源は，ノード内で一意となる固有の名称を付けてください。排他制御する資源の名称が正しいかどうかは OpenTP1 では判断できません。UAP で指定する資源名称は論理的に正しい名称を指定してください。排他指定の有効範囲は，一つの OpenTP1 システム内で，かつ同じノード内に限ります。ほかの OpenTP1 システム上の UAP との排他制御はできません。

4.6.2　排他の種類

排他制御では，OpenTP1 システム内で固有の資源名称と排他の条件（**排他制御モード**）を指定します。排他制御モードには次の 2 種類があります。

参照目的の排他（共用モード　PR　Protected Retrieve）

UAP は排他指定した資源の参照だけできます。ほかの UAP からの参照だけを許可します。

更新目的の排他（排他モード　EX　EXclusive）

UAP は排他指定した資源の参照，更新ができます。ほかの UAP からの参照，更新を禁止します。

排他を指定しようとした資源が，ほかの UAP ですでに排他指定をされていた場合，互いの排他モードの内容によって，資源を共用できる場合とできない場合があります。

一つの資源に対して複数の UAP から排他制御の指定があった場合の，共用の可否を次の表に示します。資源を共用できない場合に，エラーリターンするか，資源の解放待ちにするかは UAP で指定できます。

表 4-13　排他制御モードの組み合わせと共用の可否

資源を確保中の UAP のモード	排他要求した UAP のモード	
	参照目的の排他（PR）	更新目的の排他（EX）
参照目的の排他（PR）	共用できます。	共用できません。
更新目的の排他（EX）	共用できません。	共用できません。

4.6.3　排他待ち限界経過時間の指定

UAP から排他を指定されている資源に対して，ほかの UAP が排他要求した場合，その UAP は資源の解放を待つことができます。さらに続けて解放待ちの UAP がある場合は，ユーザサービス定義の排他待ちの優先順位の指定によって，資源の解放待ちの順番を決定して，資源の解放を待ちます。

ロックサービス定義に排他待ち限界経過時間を指定して，解放待ちの UAP が指定した時間を超えた場合は，その UAP はエラーリターンします。

UAP が排他待ちをしている資源と排他待ち限界経過時間は，lckls コマンドで知ることができます。

4.6.4　排他制御用のテーブルプール不足のとき

排他制御は，共用メモリのテーブルプールで管理されています。このテーブルプールが満杯のときは，dc_lck_get 関数はエラーリターンします。このときはサービス関数で abort()などを呼び出して，排他の処理を取り消してください。

4.6.5　排他の解除方法

排他指定した資源を解放する方法は，次の二つの場合があります。

- 資源を確保している UAP から排他を解除します。解除する資源名称を指定して排他を解除する場合は，dc_lck_release_byname 関数【CBLDCLCK('RELNAME ')】を呼び出します。UAP で確保しているすべての資源を一度に解放する場合は，dc_lck_release_all 関数【CBLDCLCK('RELALL ')】を呼び出します。排他を解除する関数は，その資源の排他を指定した UAP からだけ呼び出せます。排他を解除された資源は，OpenTP1 が資源の解放待ちの UAP に割り当てます。
- 資源を確保している UAP の同期点処理後に，その UAP が確保していたすべての資源を OpenTP1 で解放します。UAP の終了形態が正常でも異常でも，OpenTP1 で自動的に解放します。

4.6.6　ロックマイグレーション

dc_lck_get 関数で資源の排他をする場合，一つのグローバルトランザクション内の各トランザクションブランチに，資源の占有権が順次移動します。この機能を**ロックマイグレーション**といいます。ロックマイグレーションによって，トランザクションブランチ間の排他待ちやデッドロックを防げます。そのため，あるグローバルトランザクションで排他を指定した資源に対しては，資源の解放をしないかぎり，一つのグローバルトランザクション内のどのトランザクションブランチからでもアクセスできます。

ロックマイグレーションは次の場合に保証されます。

- 一つのノード内にグローバルトランザクションがある（グローバルトランザクションが複数のノードのサービスから構成されていない）場合。

(1)　ロックマイグレーションと排他制御モード

ロックマイグレーションでは，PR モードで排他をしても，別のトランザクションブランチで EX モードと指定すれば，それ以降の排他はすべて EX モードになります。一つのグローバルトランザクション内では，一度 EX モードで排他した資源には，PR モードで排他できません。すべて EX モードでの排他となります。

(2)　ロックマイグレーションでの資源の解放

ロックマイグレーションの排他は，グローバルトランザクションが終了したときに，自動的に解放されます。グローバルトランザクションの終了を待たないで，排他を解放できる場合は，次に示す方法で解放してください。

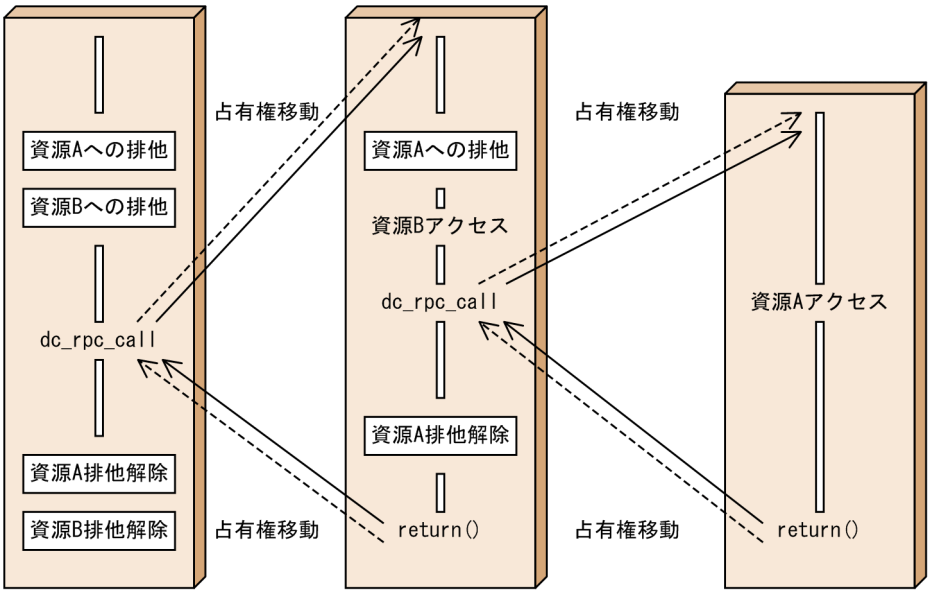
- dc_lck_release_byname 関数での解放
  ロックマイグレーションの排他をコミット／ロールバックを待たないで解放するときは，資源名称を指定して排他を解除する関数（dc_lck_release_byname 関数）を呼び出してください。排他の解除は，どのトランザクションブランチでもできます。この場合，グローバルトランザクション内でその資源に対して排他を指定した回数だけ，排他解除の関数を呼び出すまでは，資源は解放されません。
- dc_lck_release_all 関数での解放
  全資源の排他を解除する関数（dc_lck_release_all 関数）をどこかのトランザクションブランチで呼び出せば，どのトランザクションブランチで何回排他を指定していても，すべての資源が解放されます。

(3)　ロックマイグレーションでの注意事項

同期応答型 RPC の dc_rpc_call 関数でロックマイグレーションが起こったあとで，この dc_rpc_call 関数がタイムアウトなどでエラーリターンした場合は，排他した資源に対するアクセス（確保済みの資源へのアクセスや新たな排他要求）はしないでください。アクセスした場合の動作は保証しません。

ロックマイグレーションの概要を次の図に示します。

図 4-32　ロックマイグレーションの概要



4.6.7　排他のテスト

資源の排他をする関数（dc_lck_get 関数）で，資源を排他できるかどうかをテスト（flags に DC_LCK_TEST を設定）できます。この場合 dc_lck_get 関数は，資源を確保できる状態でも実際に排他をしないで正常終了します。指定した資源がほかの UAP から排他されていて共用ができない場合は，排他待ちの指定の有無に関係なく DCLCKER_WAIT(00450)でエラーリターンとなります。そのほか，テストの排他の場合は次のリターン値が戻ります。デッドロックやタイムアウト，メモリ不足のエラーリターンはしません。ロックマイグレーションは起こりません。

- DCLCKER_PARAM(00401)
  引数の設定誤り
- DCLCKER_OUTOFTRN(00455)
  トランザクション処理でない UAP からの関数呼び出し
- DCLCKER_VERSION(00457)
  OpenTP1 のライブラリとロックサービスのバージョン不一致

(1)　排他のテストをする場合の注意事項

排他のテストは，排他できる状態かどうかを知るために行います。排他のテストが正常終了したことで，それ以降の排他要求が正常終了するかどうかは保証されません。また，排他のテストが正常終了しても，実際には資源は占有されていません。そのため，排他のテストが正常終了したあとに排他を解除する関数（dc_lck_release_all 関数，dc_lck_release_byname 関数）を呼び出すとエラーリターンします。

4.7　デッドロックが起こったときの処置

OpenTP1 の UAP では，ほかの UAP と資源を共有して，並行に動作します。各 UAP は資源の内容に矛盾が起こらないように，資源を排他制御します。ただし，複数の UAP が異なる順番で複数の資源を確保しようとすると，互いが確保している資源が解放されるのを待って処理が止まってしまうことがあります。このような状態を**デッドロック**といいます。

さらに，複数の UAP が異なるリソースマネジャ（RM）にアクセスすると，DAM ファイルサービスの排他制御や TAM ファイルサービスの排他制御が互いに影響し合って，デッドロックが起こってしまう場合もあります。ここでは，デッドロックを起こさないための注意と，デッドロックが起こった場合の OpenTP1 の処置について説明します。

4.7.1　デッドロックを避けるための注意

デッドロックを避けるため，UAP からは，次のように資源へアクセスしてください。

- トランザクションの終了まで資源を占有する UAP では，できるだけ最後の処理で資源を確保します。
- 処理の途中で解放できる資源は，できるだけ早く解放します。
- 複数の資源を使用する場合は，UAP 間で資源にアクセスする順番を統一しておきます。また，一つのシステム全体で資源にアクセスする順番を統一しておきます。
- デッドロックが起こったときの，UAP の処理の優先順位を明確にしておきます。

4.7.2　デッドロック時の OpenTP1 の処置

デッドロックが起こった場合，OpenTP1 は UAP の排他待ち優先順位に従って，優先順位の低い UAP プロセスからの排他要求をエラーリターンさせます。UAP の排他待ち優先順位は，**ユーザサービス定義の deadlock_priority オペランド**に指定します。

(1)　デッドロック時の UAP の処置

デッドロックが原因で，資源を確保しようとした関数がエラーリターンした場合，UAP では次のように対処してください。

(a)　SUP，SPP でデッドロックが起こったときの処置

SUP，または SPP の処理でデッドロックが起こった場合は，ロールバックの関数（dc_trn_unchained_rollback 関数，dc_trn_chained_rollback 関数，tx_rollback 関数）でトランザクションをロールバックさせてください。デッドロックでロールバックした SUP，または SPP は再実行（リトライ）しません。もう一度該当するサービスをクライアント UAP から要求し直してください。

(b) MHP でデッドロックが起こったときの処置

MHP の処理でデッドロックが起こった場合は，dc_mcf_rollback 関数でロールバックしてください。再実行（リトライ）するかどうかは，dc_mcf_rollback 関数の引数に指定します。

(2) デッドロック情報，タイムアウト情報の出力

デッドロックが起こった場合，デッドロックの原因となった UAP の詳細情報を，ロックサービスがあるノードのディレクトリに出力できます。この情報を**デッドロック情報**といいます。

資源の解放を待っている UAP が，ロックサービス定義の lck_wait_timeout オペランドに指定した時間を超えた場合，UAP で呼び出した関数はエラーリターンします。このとき，確保しようとした資源に関する詳細情報を，ロックサービスがあるノードのディレクトリに出力できます。この情報を**タイムアウト情報**といいます。

デッドロック情報，タイムアウト情報を出力するかどうかは，ロックサービス定義の lck_deadlock_info オペランドに指定します。

デッドロック情報，タイムアウト情報の出力形式については，「付録 B　デッドロック情報の出力形式」を参照してください。

(a) デッドロック情報，タイムアウト情報の削除

デッドロック情報，タイムアウト情報を削除する方法を次に示します。

- **コマンドで削除する方法**
  lckrminf コマンドを実行します。
- **OpenTP1 の開始時に，前回までのオンラインで作成した情報を削除する方法**
  ロックサービス定義の lck_deadlock_info_remove **オペランド**と lck_deadlock_info_remove_level **オペランド**に，削除する条件を指定しておきます。

(3) 複数のリソースマネジャでデッドロックが起こった場合の OpenTP1 の処置

複数のリソースマネジャにアクセスする UAP 同士でデッドロックが起こった場合の，OpenTP1 の処置を示します。

(a) OpenTP1 で排他制御する RM（DAM，TAM）同士のデッドロックの場合

ユーザサービス定義の deadlock_priority **オペランド**に指定した，**UAP の排他待ち優先順位**の値に従って，処置します。

(b) OpenTP1 で排他制御する RM（DAM，TAM）と 他社 RM とのデッドロックの場合

ロックサービス定義の lck_wait_timeout **オペランド**に指定した**排他待ち限界経過時間**で監視します。RM 固有に指定した限界経過時間は参照しないので，ロックサービス定義の排他待ち限界経過時間は必ず指定しておいてください。

(c) 他社 RM と他社 RM とのデッドロックの場合

RM 固有に指定した限界経過時間も，ロックサービス定義の排他待ち限界経過時間も参照しません。この場合，OpenTP1 はトランザクションの限界経過時間で UAP を監視します。**ユーザサービス定義**，**ユーザサービスデフォルト定義**，**トランザクションサービス定義**の trn_expiration_time **オペランド**に指定した値を超えた場合に，該当する UAP のプロセスを異常終了させます。

5 X/Open に準拠したアプリケーションプログラミングインタフェース

X/Open に準拠したアプリケーションプログラミングインタフェース（XATMI インタフェース，TX インタフェース）を OpenTP1 のアプリケーションプログラムで使う場合の機能について説明します。

この章では，各機能を **C 言語の関数名**で説明します。C 言語の関数名に該当する COBOL 言語の API は，関数を最初に説明する個所に【 】で囲んで表記します。それ以降は，C 言語の関数名に統一して説明します。COBOL 言語の API がない関数の場合は，【 】の表記はしません。

5.1　XATMI インタフェース（クライアント／サーバ形態の通信）

XATMI インタフェースとは，オープンシステムの標準化団体である X/Open で規定する DTP モデルに準拠した，クライアント／サーバ形態の通信をするための API です。OpenTP1 では，XATMI インタフェースを使って，UAP プロセス間の通信ができます。

- **OpenTP1 の UAP 種別と XATMI インタフェースの関係**

  XATMI インタフェースの通信を使えるのは，SUP，SPP です。MHP では XATMI インタフェースの関数は使えません。また，SUP，SPP の両方に，XATMI インタフェース定義ファイルから作成したスタブを結合しておいてください。

  UAP プロセスの実行環境や開始，または終了方法など，OpenTP1 の UAP 操作に関しては，特に記述しないかぎり，OpenTP1 の RPC（dc_rpc_call 関数）を使ったクライアント／サーバ形態の通信と同様です。

5.1.1　XATMI インタフェースでできる通信形態

XATMI インタフェースでできる通信形態について説明します。

XATMI インタフェースの通信は，通信プロトコルに TCP/IP を使います。また，通信プロトコルに OSI TP を使っている場合にも，XATMI インタフェースを使えます。通信プロトコルと XATMI インタフェースの機能の関係については，「5.1.2　XATMI インタフェースの機能」を参照してください。

OSI TP 通信をする場合は，OpenTP1 システムに TP1/NET/OSI-TP-Extended が必要です。TP1/NET/OSI-TP-Extended のセットアップ手順については，マニュアル「OpenTP1 プロトコル TP1/NET/OSI-TP-Extended 編」を参照してください。

(1)　通信の形態

XATMI インタフェースでは，次に示す通信形態を提供します。

- **リクエスト／レスポンス型サービスの通信**

  サービス関数に一つの要求を送信して，一つの応答を受信する通信です。OpenTP1 のリモートプロシジャコールと同様の，サービスを要求して結果を受信する通信です。

- **会話型サービスの通信**

  通信相手のサービス関数を起動して確立したコネクションを介して，サービス関数と互いにデータを送受信する通信です。

  会話型サービスの通信ができるのは，通信プロトコルに TCP/IP を使っている場合だけです。通信プロトコルに OSI TP を使っている場合は，会話型サービスの通信はできません。

(2) サービスの要求方法

サービスを要求するときは，サーバ UAP のサービス関数を識別する名称（**サービス名**）を引数に設定した関数を呼び出します。

(3) XATMI インタフェースの通信で使うデータ

XATMI インタフェースの通信では，C 言語の場合は**構造体**，COBOL 言語の場合は**レコード**を送受信のデータに使えます。そのため，1 回のサービス要求でまとまったデータを送信できます。このデータを C 言語の場合は**型付きバッファ（タイプトバッファ）**，COBOL 言語の場合は**型付きレコード（タイプトレコード）**といいます。通信で使う型付きのデータについては，「5.1.6 通信データの型」を参照してください。

5.1.2 XATMI インタフェースの機能

XATMI インタフェースのアプリケーションプログラミングインタフェースと通信プロトコル別で使える機能について説明します。

(1) XATMI インタフェースのライブラリ関数

XATMI インタフェースのライブラリ関数の一覧を次の表に示します。

表 5-1 XATMI インタフェースのライブラリ関数の一覧

XATMI インタフェースの機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL 言語ライブラリ
リクエスト／レスポンス型サービスの通信	リクエスト／レスポンス型サービスの呼び出しと応答の受信	tpcall()	TPCALL
	リクエスト／レスポンス型サービスの呼び出し	tpacall()	TPACALL
	リクエスト／レスポンス型サービスからの非同期応答の受信	tpgetrply()	TPGETRPLY
	リクエスト／レスポンス型サービスのキャンセル	tpcancel()	TPCANCEL
会話型サービスの通信	会話型サービスとのコネクションの確立	tpconnect()	TPCONNECT
	会話型サービスとのコネクションの切断	tpdiscon()	TPDISCON
	会話型サービスからのメッセージの受信	tprecv()	TPRECV
	会話型サービスへのメッセージの送信	tpsend()	TPSEND
通信データの型の操作	型付きバッファの割り当て	tpalloc()	－
	型付きバッファの解放	tpfree()	－

XATMI インタフェースの機能		ライブラリ関数名	
		C 言語ライブラリ	COBOL 言語ライブラリ
通信データの型の操作	型付きバッファのサイズの変更	tprealloc()	－
	型付きバッファの情報の取得	tptypes()	－
サービス名を動的に管理	サービス名の広告	tpadvertise()	TPADVERTISE
	サービス名の広告の取り消し	tpunadvertise()	TPUNADVERTISE
サーバで使う関数	サービス関数のテンプレート※	tpservice()	－
	サービスルーチンの開始※	－	TPSVCSTART
	サービス関数からのリターン	tpreturn()	TPRETURN

（凡例）

－：該当する API がないことを示します。

注※

C 言語と COBOL 言語ではサービス開始を宣言する方法が異なるため，別の API としています。tpservice()は，C 言語のサービスの実体を示します。

XATMI インタフェースの関数と OpenTP1 の UAP の関係を次の表に示します。

表 5-2　XATMI インタフェースの関数と OpenTP1 の UAP の関係

XATMI インタフェースの関数	SUP		SPP			MHP		オフラインの業務をする UAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
tpacall()	○	○	○	○	○	－	－	－
tpadvertise()	－	－	○※1	○※1	○※1	－	－	－
tpalloc()	○	○	○	○	○	－	－	－
tpcall()	○	○	○	○	○	－	－	－
tpcancel()	○	○	○	○	○	－	－	－
tpconnect()	○	○	○	○	○	－	－	－
tpdiscon()	○	○	○	○	○	－	－	－
tpgetrply()	○	○	○	○	○	－	－	－
tpfree()	○	○	○	○	○	－	－	－
tprecv()	○	○	○	○	○	－	－	－
tprealloc()	○	○	○	○	○	－	－	－

XATMI インタフェースの関数	SUP		SPP			MHP		オフラインの業務をするUAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
tpreturn()	−	−	○※2	○※2	○※2	−	−	−
tpsend()	○	○	○	○	○	−	−	−
tpservice()※3	−	−	−	−	−	−	−	−
tptypes()	○	○	○	○	○	−	−	−
tpunadvertise()	−	−	○※1	○※1	○※1	−	−	−

（凡例）

○：関数を呼び出せます。

−：関数を呼び出せません。

注

MHP の「トランザクション処理の範囲でない」とは，非トランザクション属性の MHP，または MHP のメイン関数の範囲を示します。

注※1

サービス関数の中でだけ，呼び出せます。

注※2

XATMI インタフェースのサービス関数をリターンするためだけに使います。

注※3

tpservice()は，サービス関数の実体です。

(2) XATMI インタフェースの機能と通信プロトコルの関係

XATMI インタフェースの通信は，TCP/IP 通信でも OSI TP 通信でも使えます。ただし，通信プロトコルによって制限がある機能もあります。XATMI インタフェースの機能と通信プロトコルの関係を次の表に示します。

表 5-3 XATMI インタフェースの機能と通信プロトコルの関係

XATMI インタフェースの機能	通信プロトコル	
	TCP/IP	OSI TP
リクエスト／レスポンス型の通信	○	○
会話型サービスの通信	○	−
型付きのデータ送信	○	○※
OpenTP1 同士のクライアント／サーバ型通信	○	○
他 TP モニタへのトランザクション拡張	−	○

（凡例）

○：該当する通信プロトコルで使えます。

－：該当する通信プロトコルでは使えません。

注※

OSI TP を使って OpenTP1 以外のシステムとクライアント／サーバ形態で通信する場合でも，通信データの型を変換して送信できます。指定する通信データの型については，「5.1.6　通信データの型」を参照してください。

5.1.3　リクエスト／レスポンス型サービスの通信

XATMI インタフェースの，リクエスト／レスポンス型サービスの通信形態について説明します。

(1)　リクエスト／レスポンス型サービスの種類

リクエスト／レスポンス型サービスの通信形態の種類を次に示します。

(a)　同期的に応答を受信する通信

サービスを要求してから，応答が返ってくるのを待つ通信です。サービスの要求には，関数 tpcall() 【TPCALL】を使います。

（監視時間について）

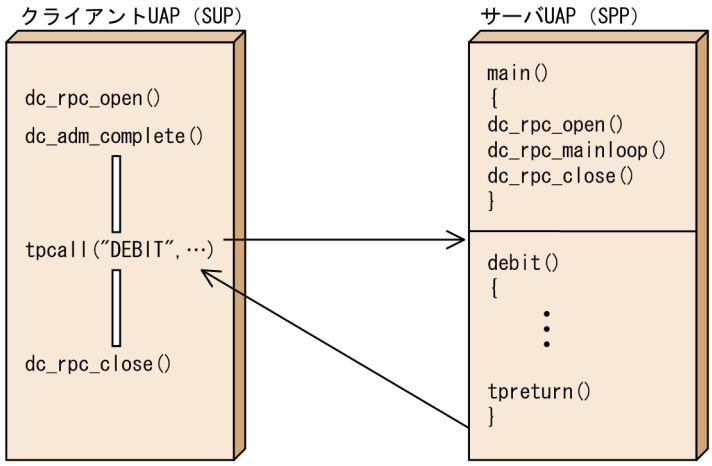
同期的に応答を受信する通信では，応答が返るまでの待ち時間を監視します。最大応答待ち時間は，OpenTP1 の定義で指定します。定義については，マニュアル「OpenTP1 システム定義」を参照してください。

tpcall()を呼び出したプロセスがトランザクション下にある場合，最大応答待ち時間は，定義の trn_expiration_time オペランドで指定した値となります。この場合，最大応答待ち時間を過ぎると，そのプロセスは異常終了します（tpcall()はエラーリターンしません）。

tpcall()を呼び出したプロセスがトランザクション下にない場合，最大応答待ち時間は，定義の watch_time オペランドで指定した値となります。この場合，最大応答待ち時間を過ぎると，tpcall()はエラーリターンします。

同期的に応答を受信するリクエスト／レスポンス型サービスの通信形態を次の図に示します。

図 5-1　同期的に応答を受信するリクエスト／レスポンス型サービスの通信形態



(b)　非同期に応答を受信する通信

サービスを要求してから，応答が返ってくるのを待たないで，処理を続ける通信です。その後，応答を受信する関数を使って，応答を受信します。サービスの要求には，関数 tpacall()【TPCALL】を使います。そして，応答を受信するための関数 tpgetrply()【TPGETRPLY】を使います。

(監視時間について)

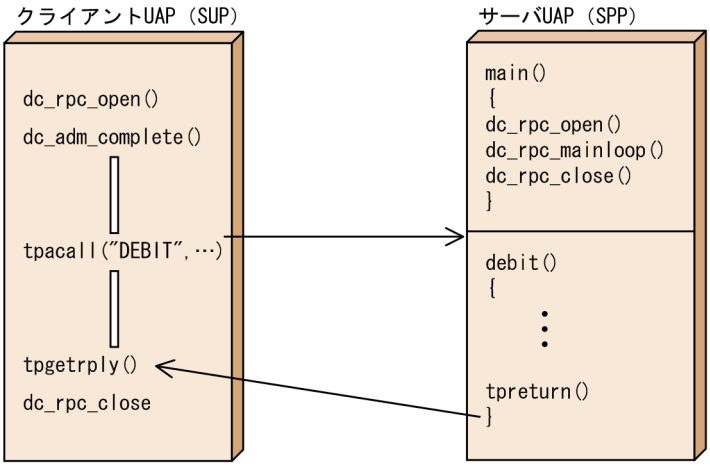
非同期に応答を受信する通信では，tpgetrply()で応答を受信するまで待ちます。最大応答待ち時間は，OpenTP1 の定義で指定します。定義については，マニュアル「OpenTP1 システム定義」を参照してください。

tpacall()，または tpgetrply()を呼び出したプロセスがトランザクション下にある場合，最大応答待ち時間は，定義の trn_expiration_time オペランドで指定した値となります。この場合，最大応答待ち時間を過ぎると，そのプロセスは異常終了します（tpgetrply()はエラーリターンしません）。

tpacall()，または tpgetrply()を呼び出したプロセスがトランザクション下にない場合，最大応答待ち時間は，定義の watch_time オペランドで指定した値となります。この場合，最大応答待ち時間を過ぎると，tpgetrply()はエラーリターンします。

非同期に応答を受信するリクエスト／レスポンス型サービスの通信形態を次の図に示します。

図 5-2　非同期に応答を受信するリクエスト／レスポンス型サービスの通信形態



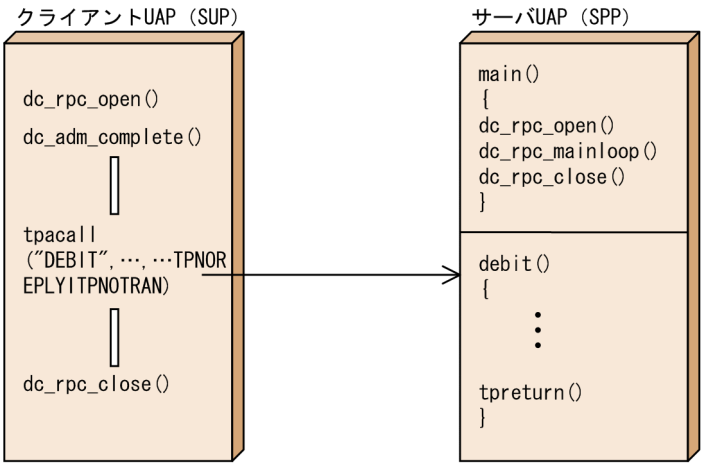
(c)　応答を返さない通信

サービス要求の処理結果が戻らない通信です。応答を返さない通信では，関数 tpacall()のフラグに TPNOREPLY を設定して使います。ただし，tpacall()のフラグに TPNOTRAN も合わせて設定する必要があります。

非応答型の通信でサービス要求をした場合，応答は受信できません。サービスを要求した UAP は，処理を続けます。

応答を返さないリクエスト／レスポンス型サービスの通信形態を次の図に示します。

図 5-3　応答を返さないリクエスト／レスポンス型サービスの通信形態



(2)　リクエスト／レスポンス型サービスの通信の時間監視

リクエスト／レスポンス型サービスの通信では，時間監視はすべて OpenTP1 の定義に指定した値に従います。定義については，マニュアル「OpenTP1 システム定義」を参照してください。

タイムアウトには，トランザクション処理がタイムアウトになったことを示す**トランザクションタイムアウト**と，ブロッキング状態の待ちによるタイムアウトによる**ブロッキングタイムアウト**があります。トランザクションタイムアウトが起こった場合，そのプロセスは異常終了します。

設定したタイムアウト値を超えて，処理を待つ場合もあります（該当のデータでない応答が返ってきても，OpenTP1 の監視タイマはリセットされるため）。また，フラグに TPNOTIME を設定した場合は，タイムアウト値を無限大とします。ただし，トランザクションタイムアウトはこのフラグの有無に関係なく起こります。

(3) リクエスト／レスポンス型サービスの通信とトランザクションの関係

OpenTP1 で使えるトランザクション管理の関数（dc_trn_～，または tx_～）でトランザクションを制御します。OpenTP1 でのトランザクション決着，または rollback_only 状態かどうかは，サービス関数の処理結果，またはトランザクションを制御する関数によって決定されます。ただし，リクエスト／レスポンス型サービスでは，次に示すエラーが起こると，該当のトランザクションブランチを rollback_only 状態とします。

- トランザクションタイムアウトが発生（プロセスは異常終了します）。
- 型付きバッファ受信のエラー（受信した型が，許可されていない）。
- TPESVCERR，TPESVCFAIL エラーの発生（tpreturn()のエラー通知，または，tpreturn()を使ったユーザからのエラー通知）。
- TPESYSTEM エラーの発生（ただし，TPESYSTEM がリターンしても rollback_only 状態とならない場合もあります）。

(a) 同期的に応答を受信する通信のトランザクション管理

呼び出し側がトランザクション下にある場合，呼び出すサービスをトランザクションとして処理するかどうかは，関数 tpcall()の引数 flags に設定したパラメタで決まります。呼び出すサービスをトランザクションとしない場合に限り，TPNOTRAN を設定してください。TPNOTRAN を設定しても，トランザクションタイムアウトは起こります。

(b) 非同期的に応答を受信する通信のトランザクション管理

呼び出し側がトランザクション下にある場合，呼び出すサービスをトランザクションとして処理するかどうかは，関数 tpacall()の引数 flags に設定したパラメタで決まります。呼び出すサービスをトランザクションとしない場合に限り，TPNOTRAN を設定してください。TPNOTRAN を設定しても，トランザクションタイムアウトは起こります。

(c) 応答を返さない通信のトランザクション管理

応答を返さない通信はトランザクションとして処理できません。関数 tpacall()の引数 flags に TPNOTRAN を必ず設定してください。

(4) ブロッキング状態が発生した場合の処置

リクエスト／レスポンス型サービスの通信関数には，ブロッキング状態の場合の処置を示す TPNOBLOCK フラグがあります。このフラグを設定した tpgetrply()は，ブロッキング状態を検知した時点でエラーリターンします。このフラグを設定しないと，ブロッキング状態が解決するまで待つか，またはタイムアウトとなります（ただし，TPNOTIME を設定していれば，ブロッキング状態が原因でのタイムアウトとはなりません）。OpenTP1 では，このフラグが有効になるのは，tpgetrply()だけです。tpcall()，tpacall()でこのフラグを設定しても無効となります。

5.1.4 会話型サービスの通信

XATMI インタフェースの，会話型サービスの通信形態について説明します。

会話型サービスの通信ができるのは，通信プロトコルに TCP/IP を使っている場合だけです。通信プロトコルに OSI TP を使っている場合は，会話型サービスの通信はできません。

(1) 会話型サービスの通信手順

会話型サービスの通信形態では，関数を呼び出して，通信相手とコネクションを確立してから通信をします。サービスを要求する手順を次に示します。

(a) コネクションの確立

クライアント UAP は，サービス関数とのコネクションを確立します。コネクションを確立するときは，tpconnect()【TPCONNECT】を使います。tpconnect()でコネクションを確立した UAP プロセスを**オリジネータ**，相手の UAP プロセスを**サブオーディネータ**といいます。

tpconnect()が正常リターンした場合には，確立したコネクションを識別する記述子が正の値で返されます。この記述子を通信する関数に設定して，通信します。

サービス関数で tpreturn()を呼び出して処理を終了すると，コネクションは切断されます。

（コネクションの制御権）

tpconnect()の引数 flags には，制御権の有無を示す TPSENDONLY，TPRECVONLY のどちらかを設定します。TPSENDONLY を設定することで，そのプロセスは制御権を得て，データの送信関数 tpsend()を使えるようになります。逆に，TPRECVONLY を設定すると，通信相手のプロセスに制御権が渡って，tpconnect()を呼び出したプロセスはデータの受信関数 tprecv()を使えるようになります。

(b) データの送信（tpsend()）

データを送信するときは，tpsend()【TPSEND】を使います。tpsend()の引数には，tpconnect()でリターンされた記述子を設定して，使うコネクションを特定します。

コネクションの制御権がない場合には，tpsend()は使えません。この場合，tpsend()はエラーリターンします。

コネクションの制御権を通信相手のプロセスに渡したい場合は，tpsend()の flags に TPRECVONLY を設定します。このフラグを設定して tpsend()を呼び出すことで，通信相手のプロセスに制御権を渡すことになります。

サブオーディネータから tpsend()でデータを送信する場合は，受信した TPSVCINFO 構造体から得た記述子を使ってください。

(c) データの受信（tprecv()）

データを受信するときは，tprecv()【TPRECV】を使います。データは，非同期に受信します。tprecv()は，コネクションの制御権を持たないプロセスからだけ呼び出せます。

（監視時間について）

フラグに TPNOBLOCK を設定していない場合，tprecv()はデータを受信するまで待ちます。最大応答待ち時間は，OpenTP1 の定義で指定します。定義については，マニュアル「OpenTP1 システム定義」を参照してください。

tprecv()を呼び出したプロセスがトランザクション下にある場合，最大応答待ち時間は，OpenTP1 のシステム定義 trn_expiration_time オペランドで指定した値となります。この場合，最大応答待ち時間を過ぎると，そのプロセスは異常終了します（tprecv()はエラーリターンしません）。

tprecv()を呼び出したプロセスがトランザクション下にない場合，最大応答待ち時間は，OpenTP1 のシステム定義 watch_time オペランドで指定した値となります。この場合，最大応答待ち時間を過ぎると，tprecv()はエラーリターンします。

(d) コネクションの切断

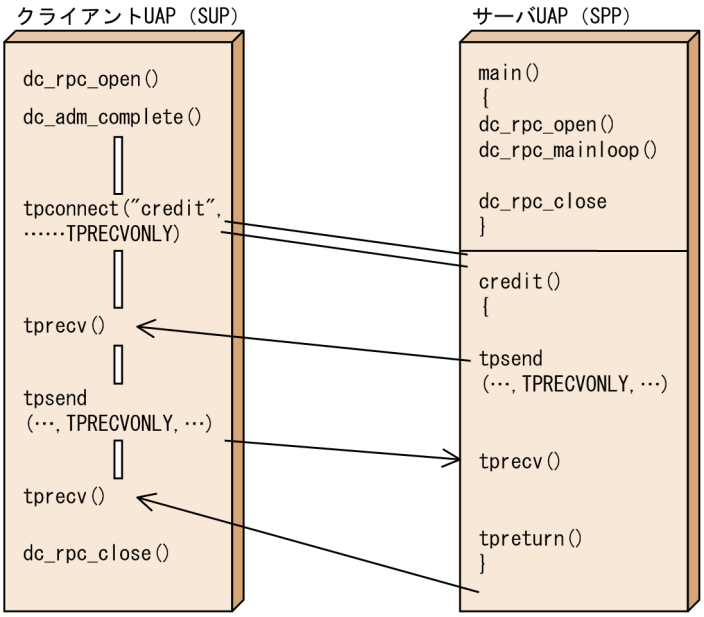
サービス関数の処理が終了して，tpreturn()【TPRETURN】を呼び出すと，コネクションは正常に切断されます。また，通信中にエラーが起こったことで，コネクションが切断される場合もあります。

(e) コネクションの強制切断

何らかの理由でコネクションを強制的に切断する場合は，tpdiscon()【TPDISCON】を使います。tpdiscon()に設定した記述子は，以降の処理では無効になります。また，トランザクションの処理の場合は，サブオーディネータ側のトランザクションブランチは rollback_only 状態になります。

会話型サービスの通信形態を次の図に示します。

図 5-4　会話型サービスの通信形態



(2)　会話型サービス通信の時間監視

会話型サービスの通信では，時間監視はすべて OpenTP1 の定義に指定した値に従います。定義については，マニュアル「OpenTP1 システム定義」を参照してください。

タイムアウトには，トランザクション処理がタイムアウトになったことを示す**トランザクションタイムアウト**と，ブロッキング状態の待ちによるタイムアウトによる**ブロッキングタイムアウト**があります。トランザクションタイムアウトが起こった場合，そのプロセスは異常終了します。

設定したタイムアウト値を超えて，処理を待つ場合もあります（該当のデータでない応答が返ってきても，OpenTP1 の監視タイマはリセットされるため）。また，フラグに TPNOTIME を設定した場合は，タイムアウト値を無限大とします。ただし，トランザクションタイムアウトはこのフラグの有無に関係なく起こります。

(3)　会話型サービスの通信とトランザクションの関係

OpenTP1 で使えるトランザクション管理の関数（dc_trn_～，または tx_～）でトランザクションを制御します。OpenTP1 でトランザクションを決着するかどうか，または rollback_only 状態かどうかは，サービス関数の処理結果，またはトランザクションを制御する関数によって決定されます。ただし，会話型サービスでは，次に示すエラーが起こると，該当のトランザクションブランチを rollback_only 状態とします。

- トランザクションタイムアウトが発生（異常終了します）。
- 型付きバッファ受信のエラー（受信した型が，許可されていない）。
- TPESYSTEM エラーの発生（ただし，TPESYSTEM がリターンしても rollback_only 状態とならない場合もあります）。
- tpdiscon()を実行。

- tpsend()でエラーコード TPEEVENT（イベントコードが TPEV_SVCERR，TPEV_SVCFAIL）。
- tprecv()でエラーコード TPEEVENT（イベントコードが TPEV_DISCONIMM，TPEV_SVCERR，TPEV_SVCFAIL）。

呼び出し側がトランザクション下にある場合，呼び出すサービスをトランザクションとして処理するかどうかは，関数 tpconnect()の引数 flags に設定したパラメタで決まります。呼び出すサービスをトランザクションとしない場合に限り，TPNOTRAN を設定してください。TPNOTRAN を設定しても，トランザクションタイムアウトは起こります。

(4) ブロッキング状態が起こった場合の処置

会話型サービスの通信関数には，ブロッキング状態の場合の処置を示す TPNOBLOCK フラグがあります。このフラグを設定した tprecv()は，ブロッキング状態を検知した時点でエラーリターンします。このフラグを設定しないと，ブロッキング状態が解決するまで待つか，またはタイムアウトとなります（ただし，TPNOTIME を設定していれば，ブロッキング状態が原因でのタイムアウトとはなりません）。
OpenTP1 では，このフラグが有効になるのは，tprecv()だけです。tpconnect()，tpsend()でこのフラグを設定しても無効となります。

(5) イベントの受信

コネクションを識別する記述子 cd にイベントが存在した場合，会話型サービスの通信関数（tpsend()，tprecv()）でそのイベントを受信できます。イベントは，通信に関する情報を示します。詳細については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。

5.1.5 OpenTP1 での注意事項

OpenTP1 で XATMI インタフェースを使って通信する場合は，次の点に注意してください。

- XATMI インタフェースを使うユーザサーバのユーザサービス定義には，次の値を必ず指定してください。

```
server_type = ”xatmi”
```

- XATMI インタフェースでは，サービスグループの概念はありません。ただし，OpenTP1 で XATMI インタフェースを使った通信をする場合は，UAP のユーザサービス定義にサービスグループを設定してください。
- XATMI インタフェースを使う場合は，ユーザサービス定義，またはユーザサービスデフォルト定義に次の値を必ず指定してください。

```
trn_expiration_time = 0以外の値
trn_expiration_time_suspend = Y
```

- tpcall()，tpacall()，tpconnect()，tpsend()でデータを送信するときにブロッキング状態が起こって，一定時間が経ってもブロッキング状態が解除されなかった場合，TPENOBLOCK フラグの有無に関係

なく，TPESYSTEM がリターンされます。TPESYSTEM でエラーリターンするまでの時間は，定義のサービス要求送信リトライ回数，間隔によって決まります。

- トランザクションタイムアウトが起こったときは，TPETIME はリターンされないで，そのプロセスは異常終了します。
- dc_rpc_call 関数で呼ばれて，XATMI インタフェースの関数（tpcall()など）を呼び出す UAP には，RPC インタフェース定義と XATMI インタフェース定義のクライアント用定義の両方を指定して作成したスタブをリンケージしておいてください。この場合のスタブを作成する方法については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。
- tx_commit()などでトランザクションを決着させた場合には，それ以前のすべての未受信データは無効となります。
- ノード間負荷バランス機能およびノード間負荷バランス拡張機能を使う場合，dc_rpc_call 関数の場合と同様に，複数用意した SPP のサービスグループ名をユーザサービス定義で一致させる必要があります。その際には，ユーザサービス定義でサービス名と実行形式ファイル名を一致させてください。サービス名が一致していない場合は，tpcall()，tpacall()，tpconnect()が失敗する場合があります。実行形式ファイル名が一致していない場合は，どのサーバ UAP がスケジュールされたかによって処理結果がまちまちになります。
- OSI-TP 通信を使用したトランザクションブランチは，相手システムとのアソシエーションがなければ回復できません。また，相手システムに対する着呼のアソシエーションだけでは回復できない場合があります。必ず発呼のアソシエーションを定義してください。トランザクションブランチの回復ができない場合は，相手システムとのアソシエーションが確立されているかどうか確認してください。ただし，同じ相手システムに対する複数のトランザクションブランチの回復が並列に実行されないで時間が掛かる場合があります。
- XATMI インタフェース API で送受信できる最大データ長は 500 キロバイトです。

5.1.6 通信データの型

XATMI インタフェースの通信では，1 回のサービス要求でまとまったデータを送信できるように，C 言語の場合は**構造体**，COBOL 言語の場合は**レコード**を送受信のデータに使えます。このデータを，C 言語の場合は**型付きバッファ（タイプトバッファ）**，COBOL 言語の場合は**型付きレコード（タイプトレコード）**といいます。

(1) 通信データの型の構成

通信データの型は，**タイプ（type）**と**サブタイプ（subtype）**から構成されます。UAP で使う通信データの型は，UAP を作成するときにスタブのソースファイル（**XATMI インタフェース定義**）で指定します。XATMI インタフェース定義については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。

(a) タイプ（type）

XATMI インタフェースで規定するデータ型の呼称です。それぞれのタイプには次の特長があります。

- X_OCTET
  オクテットの配列（バイト列）から成るデータ型です。X_OCTET には，サブタイプはありません。
- X_COMMON
  入れ子にならない構造体またはレコードのことです。サブタイプで構造を特定します。
- X_C_TYPE
  入れ子にならない構造体またはレコードのことです。サブタイプで構造を特定します。

(b) サブタイプ（subtype）

それぞれのタイプで使える範囲のデータを要素に持つ，構造体またはレコードを示す名称です。

タイプで使えるデータ型については，「5.1.6(3) タイプで使えるデータ型の一覧」を参照してください。

(2) 通信データの型の使い方

型付きバッファまたは型付きレコードを使うと，C 言語の場合は構造体で，COBOL 言語の場合はレコードでデータをやり取りできます。また，関数のフラグの設定によって，設定した受信用のデータ型と異なるタイプやサブタイプ，異なる大きさのデータでも受信できます。ただし，UAP で扱える通信データの型は，UAP に XATMI インタフェース定義で事前に定義した値と一致していることが前提です。

(3) タイプで使えるデータ型の一覧

型付きバッファを使うときは，XATMI インタフェース定義（サーバ UAP 用）にタイプ，サブタイプ，およびデータ型を指定します。XATMI インタフェース定義ファイルからスタブを生成して，スタブのオブジェクトファイルをサーバ UAP にリンケージすると，該当する型付きバッファを使えるようになります。XATMI インタフェース定義については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編を参照してください。

通信プロトコルに OSI TP を使って OpenTP1 以外のシステムと通信する場合でも，通信相手システムで型付きバッファおよび型付きレコードを認識できるように変換して送信できます。

タイプで使えるデータ型の一覧を次の表に示します。識別子とは XATMI インタフェース定義に記述するデータ型を示し，C 言語のデータ型および COBOL 言語のデータとは実際にスタブに定義される型付きバッファおよび型付きレコードを示します。OpenTP1 以外のシステムと通信するためにデータ型を変換する場合は，変換する識別子を XATMI インタフェース定義に指定します。

表 5-4　タイプで使えるデータ型の一覧

タイプ	識別子	C 言語のデータ型	COBOL 言語のデータ型	通信プロトコル		備考
				TCP/IP	OSI TP	
X_OCTET	―※1	―※1	―※1	○	○	なし
X_COMMON	short a	short a	PIC S9(4) COMP-5	○	○	なし
	short a[n]	short a[n]	PIC S9(4) COMP-5 OCCURS n TIMES	○	○	なし
	long a	DCLONG a	PIC S9(9) COMP-5	○	○	なし
	long a[n]	DCLONG a[n]	PIC S9(9) COMP-5 OCCURS n TIMES	○	○	なし
	char a※2	char a	PIC X	○	○	無変換配列
	octet a	char a	PIC X	○	○	無変換配列
	tchar a	char a	PIC X	－	○	変換配列
	char a[n]※2	char a[n]	PIC X(n)	○	○	無変換配列
	octet a[n]	char a[n]	PIC X(n)	○	○	無変換配列
	tchar a[n]	char a[n]	PIC X(n)	－	○	変換配列
X_C_TYPE	short a	short a	－	○	×	なし
	short a[n]	short a[n]	－	○	×	なし
	long a	DCLONG a	－	○	×	なし
	long a[n]	DCLONG a[n]	－	○	×	なし
	int4 a	DCLONG a	－	○	×	なし
	int4 a[n]	DCLONG a[n]	－	○	×	なし
	char a ※2	char a	－	○	×	なし
	octet a	char a	－	○	×	なし
	tchar a	char a	－	○	×	なし
	char a[n]※2	char a[n]	－	○	×	なし
	octet a[n]	char a[n]	－	○	×	なし
	tchar a[n]	char a[n]	－	○	×	なし
	float a	float a	－	○	×	なし

タイプ	識別子	C 言語のデータ型	COBOL 言語のデータ型	通信プロトコル		備考
				TCP/IP	OSI TP	
X_C_TYPE	float a[n]	float a[n]	−	○	×	なし
	double a	double a	−	○	×	なし
	double a[n]	double a[n]	−	○	×	なし
	octet a[n][n]	char a[n][n]	−	○	×	なし
	tchar a[n][n]	char a[n][n]	−	○	×	なし
	str a[n]	char a[n]	−	○	×	なし
	str a[n][n]	char a[n][n]	−	○	×	なし
	tstr a[n]	char a[n]	−	○	×	なし
	tstr a[n][n]	char a[n][n]	−	○	×	なし

（凡例）

○：該当する通信プロトコルで使えます。

×：該当する通信プロトコルでは使えません。

−：変換の識別子でも，無変換としてそのまま処理されます。

注※1

X_OCTET は，定義しなくても自動的に認識されます。XATMI インタフェース定義に X_OCTET を指定した場合は，スタブを生成するコマンドを実行したときにエラーになります。

注※2

この識別子も使えますが，新規で作成する場合は次に示す識別子を使うことをお勧めします。

X_COMMON の場合：octet または tchar

X_C_TYPE の場合：str または tstr

(4)　型付きバッファを操作する関数の使い方

XATMI インタフェースの関数で通信データを操作する方法について説明します。通信データを操作できる API は，**C 言語**でだけ使えます。COBOL 言語の場合は，通信データを操作するための API はありません。

(a)　型付きバッファの確保

型付きバッファを確保する場合は，タイプとサブタイプの値を設定した **tpalloc()**を UAP から呼び出します。tpalloc()で確保した領域は，NULL でクリアされます。

(b)　型付きバッファの再確保

型付きバッファの長さを大きくする場合は，**tprealloc()**を使います。tprealloc()で使えるバッファ型は，X_OCTET だけです。それ以外のバッファ型を指定した場合は，エラーリターンします。変更後のバッファ長が短い場合は，バッファに入り切らない分は切り捨てられます。逆に，変更後のバッファ長が長い場合は，余った部分が NULL クリアされます。

再確保に失敗すると，変更前のバッファ型も無効になります。

(c) 型付きバッファの解放

確保した領域を解放する場合は，tpfree()を使います。引数には型付きバッファへのポインタを設定します。型付きバッファのポインタでない値を設定した場合は，無視されます。

(d) 型付きバッファの情報の取得

バッファのタイプなどの情報を取得する場合は，tptypes()を使います。

(e) 型付きバッファを操作する場合の注意事項

型付きバッファを操作する関数は，C ライブラリの malloc()，realloc()，free()と一緒には使わないでください（例えば，tpalloc()で割り当てたバッファは，free()で解放しないでください）。確保した型付きバッファに対して，free()を呼び出した場合の動作は保証しません。

(5) X_OCTET 型を使用する場合の注意

X_OCTET 型の型付きバッファを使う場合は，ほかの型のバッファを使う場合と次に示す点で異なります。

1. subtype（構造体）を持たない（subtype ごとに必要な情報がない）。
2. データを変換しないで送信する（単なるビット配列として，データを扱う）。
3. 長さを示すパラメタを指定する必要がある。

5.1.7 サーバ UAP の作成方法

XATMI インタフェースの通信で使うサービス関数（サーバ UAP）から関数を呼び出す方法について説明します。OpenTP1 のサービス関数とは，コーディングの方法が異なります。

(1) サーバ UAP で提供するサービス関数のコーディング

C 言語の場合は，サービス関数のコーディングをするときは，tpservice()で示す形式に従ってください。tpservice()では，コーディングするための標準的な形式を示します。

COBOL 言語の場合は，サービスプログラムの処理で XATMI インタフェースの API を使う前に TPSVCSTART を呼び出します。

(a) サービス関数の終了方法

サービス関数は，tpreturn()【TPRETURN】を呼び出すことでその処理が終了したことになります。XATMI インタフェースの通信では，return でサービス関数を終了する直前に，必ず tpreturn()を呼び出してください。tpreturn()を呼び出したあとで何らかの処理をした場合，その動作については保証しません。

(b) サービス名の広告

サーバ UAP では，自分のサービス名がサービスを提供できる状態であることを宣言できます（サービス名の広告）。サービス名を広告する場合は，tpadvertise()【TPADVERTISE】を使います。

サーバ UAP の起動時には，ユーザサービス定義に指定してあるサービスが広告済みとなります。ユーザサービス定義に指定してあるサービス名で tpadvertise()を呼び出す必要はありません。

サービス名の広告を取り消す場合は，tpunadvertise()【TPUNADVERTISE】を使います。tpunadvertise()を呼び出すことで，該当のサービス名に対するサービス要求はエラーリターンします。いったん tpunadvertise()で広告を取り消したサービス名でも，そのあとで tpadvertise()を呼び出せば，再びサービス要求を受け付けることができます。

tpadvertise()と tpunadvertise()は，SPP でだけ使えます。ただし，dc_rpc_mainloop 関数を呼び出す前には使えません。

tpadvertise()は，広告する関数アドレスを引数に持っていて，広告できるかどうかをチェックします。OpenTP1 では，tpadvertise()を呼び出すサーバへのサービスグループと，サービスを広告しているサーバのサービスグループが同じであれば，すでに広告されていると見なして正常リターンします。サービスグループが一致しないと，tpadvertise()はエラーリターンします。

(2) 一つのノードでの負荷分散（マルチサーバ負荷バランス）と，tpunadvertise()の関係

一つのノードで負荷分散している場合（マルチサーバ負荷バランス機能を使用している場合），tpunadvertise()をどれか一つのプロセスから呼び出すと，負荷分散しているプロセスすべてでサービスを受け付けられなくなります。その後，tpadvertise()で再びサービスを広告すれば，サービス要求を受け付けられるようになります。

マルチサーバを使えるのは，キュー受信型サーバ（スケジュールサービスでスケジュールされる SPP）です。ソケット受信型サーバはマルチサーバを使えません。

5.1.8 OpenTP1 の機能と XATMI インタフェースの関係

(1) UAP トレース

XATMI インタフェースの通信を使った場合にも，UAP トレースは取得します。UAP トレースについては，マニュアル「OpenTP1 テスタ・UAP トレース 使用の手引」を参照してください。

(2) 統計情報

システム稼働統計情報の取得は，RPC の情報に追加されます。ただし，該当バージョンの OpenTP1 では，一部の障害情報は取得されません（XATMI 独自の障害や，ユーザサーバの実行時間など）。特に，会

話型サービスの形態は XATMI インタフェースの通信独自の通信方法なので，tpsend()と tprecv()などの障害の統計情報は取得できません。

(3) RPC トレース

RPC トレースも取得できます。ただし，該当バージョンの OpenTP1 では，一部の障害情報は取得されません（XATMI インタフェース独自の障害など)。特に，会話型サービスの形態は XATMI インタフェースの通信独自の方法なので，tpsend()と tprecv()などについての RPC トレースは取得できません。

(4) オンラインテスタ

XATMI インタフェースを使った UAP をテストする場合，使えないオンラインテスタの機能があります。オンラインテスタの機能と XATMI インタフェースの関係を次の表に示します。

表 5-5 オンラインテスタの機能と XATMI インタフェースの関係

オンラインテスタの機能	XATMI インタフェースで使う通信プロトコル	
	TCP/IP	OSI TP
クライアント UAP シミュレート機能	○	×
サーバ UAP シミュレート機能	○	×
MCF シミュレート機能	−	−
資源更新処理無効化機能	○	○
運用コマンドシミュレート機能	−	−
テスタファイル作成・編集機能	○	×
UAP トレース情報取得機能	○	○
UAP トレース情報マージ・編集機能	○	○
送信メッセージ編集機能	−	−
デバッガ連動機能	○	×

（凡例）

○：該当する通信プロトコルで使えます。

×：該当する通信プロトコルでは使えません。

−：XATMI インタフェースの通信とは関係しません。

5.2.1 OpenTP1 で使える TX インタフェース

X/Open に準拠した API（TX_関数）を，OpenTP1 の UAP で使えます。TX_関数でトランザクション制御をする UAP では，X/Open に準拠した仕様を持つ他社 RM を使えます。

(1) OpenTP1 の UAP と TX_関数の関係

OpenTP1 の UAP で使える TX_関数を表 5-6 に，OpenTP1 の UAP と TX_関数との関係を表 5-7 に示します。

表 5-6 OpenTP1 の UAP で使える TX_関数

TX_関数名		機能
C 言語	COBOL 言語	
tx_begin	TXBEGIN	トランザクションの開始
tx_close	TXCLOSE	リソースマネジャ集合のクローズ
tx_commit	TXCOMMIT	トランザクションのコミット （連鎖モード：TX_CHAINED 指定，非連鎖モード：TX_UNCHAINED 指定）
tx_info	TXINFORM	現在のトランザクションに関する情報の返却
tx_open	TXOPEN	リソースマネジャ集合のオープン
tx_rollback	TXROLLBACK	トランザクションのロールバック （連鎖モード：TX_CHAINED 指定，非連鎖モード：TX_UNCHAINED 指定）
tx_set_commit_return	TXSETCOMMITRET	commit_return 特性の設定
tx_set_transaction_control	TXSETTRANCTL	transaction_control 特性の設定
tx_set_transaction_timeout	TXSETTIMEOUT	transaction_timeout 特性の設定

表 5-7 OpenTP1 の UAP と TX_関数との関係

TX_関数名	SUP		SPP			MHP		オフラインの業務をする UAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
tx_begin	○	−	○	−	−	−	−	−

TX_関数名	SUP		SPP			MHP		オフラインの業務をするUAP
	トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	トランザクションの処理の範囲でない	トランザクション範囲		トランザクションの処理の範囲でない	トランザクションの処理範囲（ルート）	
				ルート	ルート以外			
tx_close	○	−	○	−	−	−	−	−
tx_commit TX_CHAINED 指定	−	○	−	○	−	−	−	−
tx_commit TX_UNCHAINED 指定	−	○	−	○	−	−	−	−
tx_info	○	○	○	○	○	−	−	−
tx_open	○	−	○	−	−	−	−	−
tx_rollback TX_CHAINED 指定	−	○	−	○	−	−	−	−
tx_rollback TX_UNCHAINED 指定	−	○	−	○	○	−	−	−
tx_set_commit_return	○	○	○	○	○	−	−	−
tx_set_transaction_control	○	○	○	○	○	−	−	−
tx_set_transaction_timeout	○	○	○	○	○	−	−	−

（凡例）

○：該当する条件で使えます。

−：使えません。

5.2.2　TX_関数の使用方法

(1)　トランザクションの開始

TX_関数でトランザクションを開始させるときは，UAP からは次のように関数を呼び出してください。この順で関数を呼び出すと，ユーザサービス定義の atomic_update オペランドの指定に関係なく，トランザクションを開始できます。また，ユーザサービス定義で atomic_update オペランドに Y を指定していれば，tx_open()【TXOPEN】と tx_close()【TXCLOSE】を呼び出さなくても，トランザクション処理を開始できます。

SUP でトランザクションを開始する場合

```
dc_rpc_open()
tx_open()
tx_begin()
```

```
      :
      :
tx_commit()（同期点処理）
tx_close()
      :   …この間で，再びtx_open()とtx_close()を呼び出せます。
dc_rpc_close()
```

SPP でトランザクションを開始する場合

（サービス関数でトランザクションを開始）

```
dc_rpc_open()
tx_open()
dc_rpc_mainloop()
      :
（サービス関数の処理）
tx_begin()
      :
      :
tx_commit()（同期点処理）
      :
（メイン関数のdc_rpc_mainloop()リターン）
tx_close()
      :
dc_rpc_close()
```

SPP のサービス関数内でトランザクションを開始させる場合には，dc_rpc_mainloop 関数を呼び出す前に，tx_open()を呼び出してください。

(2) 同期点の取得

tx_begin()【TXBEGIN】で開始したトランザクション処理は，同期点を取得する関数（tx_commit()【TXCOMMIT】，または tx_rollback()【TXROLLBACK】）で必ず完了させてください。

tx_commit()と tx_rollback()には，**連鎖モード**（TX_CHAINED）と**非連鎖モード**（TX_UNCHAINED）があります。同期点の取得後に新しいグローバルトランザクションを開始する場合は連鎖モード，開始しないでトランザクション処理を終了させる場合は非連鎖モードを設定します。連鎖モード／非連鎖モードは transaction_control 特性として設定します。transaction_control 特性は，tx_set_transaction_control()【TXSETTRANCTL】で設定します。

(3) トランザクション特性の設定

TX_関数を呼び出すことで，トランザクション処理の特性を設定できます。

(a) commit_return 特性

トランザクションを 2 相コミットするときに，1 相目が完了したときにリターンするか，または 2 相目まで完了してからリターンするかを設定できます。ただし，OpenTP1 の該当バージョンでは 2 相目まで完了しないうちにリターンする設定はできません。設定した場合はエラーリターンします。commit_return 特性は，tx_set_commit_return()【TXSETCOMMITRET】で設定します。

(b) transaction_control 特性

同期点（tx_commit()，tx_rollback()）のあとで，新しいトランザクションを開始させるかどうか（連鎖モードか非連鎖モードか）を設定します。transaction_control 特性は，tx_set_transaction_control()【TXSETTRANCTL】で，TX_CHAINED または TX_UNCHAINED のどちらかを設定します。

(c) transaction_timeout 特性

トランザクションの監視時間を設定できます。transaction_timeout 特性は，tx_set_transaction_timeout()【TXSETTIMEOUT】で設定します。システム定義の trn_expiration_time の値よりも，tx_set_transaction_timeout()で設定した transaction_timeout 特性が優先されます。

(4) トランザクションの情報取得

tx_info()【TXINFORM】で，トランザクションブランチ ID や，トランザクション特性を格納されている構造体を参照できます。

参照できる構造体の形式を次に示します。

```
XID                 xid;
COMMIT_RETURN       when_return;
TRANSACTION_CONTROL transaction_control;
TRANSACTION_TIMEOUT transaction_timeout;
TRANSACTION_STATE   transaction_state;
```

(5) ユーザサービス定義との関連

tx_open()，tx_close()を使うと，ユーザサービス定義の atomic_update オペランドの値に関係なく，tx_begin()以降の処理をトランザクションとして処理できます。

ただし，tx_begin()を呼び出した UAP からのサービス要求先でトランザクションとして処理する場合は，呼び出し先のサーバ UAP の atomic_update オペランドに Y を指定してください。

5.2.3 TX_関数を使用する場合の制限

(1) TX_関数と OpenTP1 の関数を使う場合

OpenTP1 の関数は，TX_関数と共用できます。ただし，OpenTP1 のトランザクション制御の関数（dc_trn_～）と TX_関数は併用しないでください。トランザクション制御は必ずどちらか一方の機能で統一してください。

(2) dc_rpc_open 関数と tx_open()の関係

dc_rpc_open 関数は，必ず tx_open()よりも先に呼び出してください。dc_rpc_open 関数を呼び出さないで tx_open()を呼び出した場合は，TX_ERROR でエラーリターンします。

(3) dc_rpc_close 関数と tx_close()の違い

dc_rpc_close 関数を呼び出すと，それ以降は dc_rpc_open 関数は呼び出せませんが，tx_open()は tx_close()を呼び出したあとでも再び呼び出せます。トラフィックの関係で，一度 UAP を休眠状態にしたい場合などは，一度 tx_close()を呼び出してから，再び tx_open()を呼び出します。

(4) tx_open()，tx_close()と RM 固有のオープン，クローズの関係

tx_open()と tx_close()は各 RM に対して UAP からアクセスがあること，または終了したことを知らせる関数です。tx_open()または tx_close()を呼び出すことで，各 RM は UAP から処理要求が来ることを知ります。

それに対して，RM 固有のオープン，クローズ関数（例えば，dc_dam_open 関数，dc_dam_close 関数）は，実際の処理の開始と終了を意味する関数です。tx_open()や tx_close()を呼び出しても，RM 固有のオープン，クローズ関数が不要になる訳ではありません。

5.2.4 OpenTP1 のトランザクション制御関数（dc_trn_～）との比較

(1) TX_関数と OpenTP1 のトランザクション制御関数（dc_trn_～）との対応

OpenTP1 のトランザクション制御関数（dc_trn～）と TX_関数の関係を次の表に示します。

表 5-8 OpenTP1 のトランザクション制御関数（dc_trn～）と TX_関数の関係

TX_関数名	OpenTP1 のトランザクション制御関数（dc_trn～）
tx_begin()	dc_trn_begin 関数
tx_close()	対応関数なし
tx_commit()（TX_CHAINED 指定）	dc_trn_chained_commit 関数
tx_commit()（TX_UNCHAINED 指定）	dc_trn_unchained_commit 関数
tx_info()	dc_trn_info 関数
tx_open()	対応関数なし
tx_rollback()（TX_CHAINED 指定）	dc_trn_chained_rollback 関数
tx_rollback()（TX_UNCHAINED 指定）	dc_trn_unchained_rollback 関数
tx_set_commit_return()	対応関数なし
tx_set_transaction_control()	対応関数なし
tx_set_transaction_timeout()	対応関数なし

(2) TX_関数の時間監視

TX_関数では，トランザクションの経過時間を tx_set_transaction_timeout()で監視できます。この場合，システム定義の trn_expiration_time オペランドの値よりも，tx_set_transaction_timeout()で設定した transaction_timeout 特性が優先されます。

(a) 時間監視の範囲

tx_begin()から同期点（tx_commit()，tx_rollback()）までの時間監視では，トランザクション内で呼び出した dc_rpc_call 関数がリターンするまでの時間を含めるか含めないかを選択できます。トランザクションの監視時間の範囲は，ユーザサービス定義，ユーザサービスデフォルト定義，トランザクションサービス定義の trn_expiration_time_suspend オペランドで指定できます。trn_expiration_time_suspend オペランドに指定する値とトランザクションの時間監視の詳細については，マニュアル「OpenTP1 システム定義」を参照してください。

6 X/Open に準拠したアプリケーション間通信（TxRPC）

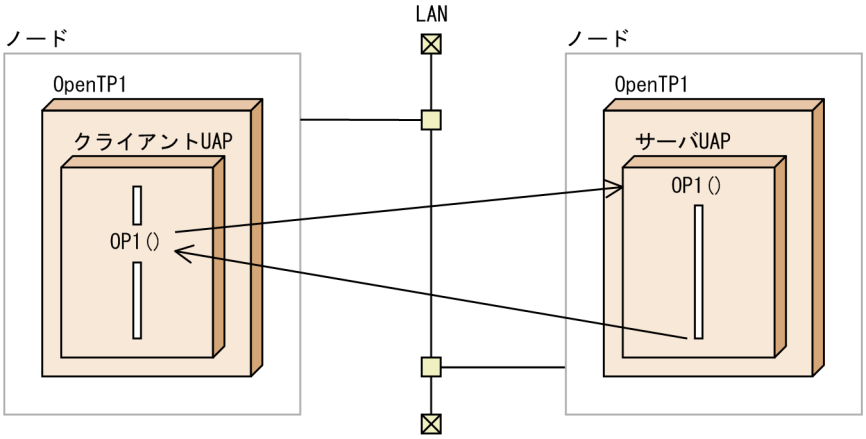
X/Open に準拠したアプリケーション間の通信方法（TxRPC インタフェース）を OpenTP1 のアプリケーションプログラムで使う場合の機能について説明します。

6.1　TxRPC インタフェースの通信

TxRPC インタフェースとは，X/Open で規定するクライアント／サーバ形態の通信方法です。OpenTP1 の UAP プロセス間で，TxRPC インタフェースの方式で通信できます。ほかのクライアント／サーバ形態の通信と異なり，TxRPC の通信では**ユーザが作成した関数**をクライアントから呼び出して通信します。ユーザが関数を作成するときは，下位の通信プロトコルを意識する必要はありません。

TxRPC インタフェースの通信の概要を次の図に示します。

図 6-1　TxRPC インタフェースの通信の概要



6.1.1　TxRPC 通信の種類

TxRPC の通信方法には，DCE RPC を使うかどうかで次に示す 2 種類があります。

- IDL-only TxRPC
- RPC TxRPC

(1)　IDL-only TxRPC

IDL コンパイラから生成されるファイルだけを使って UAP を作成する方式です。IDL-only TxRPC の場合は，DCE を前提としません。

(2)　RPC TxRPC

通信プロトコルに DCE RPC を使う方式です。このバージョンでは RPC TxRPC をサポートしていません。

6.1.2　作成できるアプリケーションプログラム

TxRPC の通信では，次の UAP を作成できます。

- クライアント UAP（SUP）
- サーバ UAP（SPP）

(1) プロセスタイプ

txidl コマンドのオプションには，作成する UAP のプロセスの型を付けるため，次に示すオプションを付けます。これを**プロセスタイプ**といいます。

- ndce：
  DCE を使わないプロセスを示します。SUP または SPP の場合に指定します。

txidl コマンドの文法については，マニュアル「OpenTP1 プログラム作成リファレンス C 言語編」を参照してください。

6.1.3 前提となるライブラリ

TxRPC 通信で，前提となるライブラリを次に示します。

SUP または SPP を作成する場合

次に示す製品がシステムに組み込んであることが前提です。

- TP1/Server Base

6.2　アプリケーションプログラムでできる通信

TxRPC では，次に示す通信ができます。

- **同期応答型のトランザクショナル RPC**
- **同期応答型の非トランザクショナル RPC**

6.2.1　TxRPC のリモートプロシジャコール

TxRPC のリモートプロシジャコールは，ユーザが作った関数を呼び出す形式です。使えるリモートプロシジャコールの形態は，同期応答型 RPC だけです。同期応答型 RPC 以外のリモートプロシジャコール（非同期応答型 RPC，非応答型 RPC，会話型 RPC など）は使えません。

TxRPC の通信では，呼び出される関数を**マネジャ**といいます。

OpenTP1 のリモートプロシジャコール（dc_rpc_call 関数）と TxRPC の通信を併用することもできます。

6.2.2　TxRPC のトランザクション処理

TxRPC の UAP では，トランザクション処理ができます。UAP の処理からトランザクションを制御する場合は，TX_関数（tx_begin()，tx_rollback()など）を使います。TX_関数のトランザクション制御については，「5.2　TX インタフェース（トランザクション制御）」を参照してください。

(1)　トランザクションが有効な範囲

OpenTP1 の TxRPC の場合，IDL-only TxRPC でトランザクション処理を使えます。

(2)　トランザクション属性

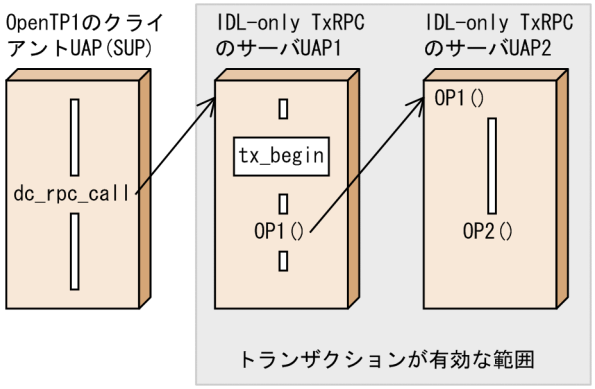
トランザクション処理をする TxRPC の UAP のプロセスには，トランザクション属性を指定します。TxRPC には次に示すトランザクション属性があります。

- transaction_mandatory
  トランザクションを必ず拡張する属性です。トランザクションでない処理からこの属性を指定した UAP を呼び出すと，エラーになります。
- transaction_optional
  トランザクションの処理から呼び出された場合には，トランザクションとして処理します。トランザクションでない処理から呼び出された場合には，トランザクションでない処理になります。

UAP のトランザクション属性は，インタフェース定義言語ファイル（IDL ファイル）に指定します。指定できるのは，transaction_mandatory か transaction_optional のどちらか片方だけです。

アプリケーションプログラムでできる通信を次の図に示します。

図 6-2　アプリケーションプログラムでできる通信



6.2.3　OpenTP1 の機能を使うアプリケーションプログラムと TxRPC のアプリケーションプログラムの関連

OpenTP1 のリモートプロシジャコール（dc_rpc_call 関数）を使う SUP，SPP と，TxRPC を使うサーバ UAP では，次に示す形式で通信できます。

- dc_rpc_call 関数で呼ばれた SPP は，TxRPC を使って別の SPP を呼び出せます。
- TxRPC のサーバ UAP から dc_rpc_call 関数を使って，SPP へサービスを要求できます。ただし，サーバ UAP のプロセスタイプが nbet の場合は，dc_rpc_call 関数は使えません。

6.3 TxRPC 通信のアプリケーションプログラムを作成する手順

TxRPC の通信を使う UAP の作成手順について説明します。

6.3.1 IDL-only TxRPC 通信の UAP を作成する手順

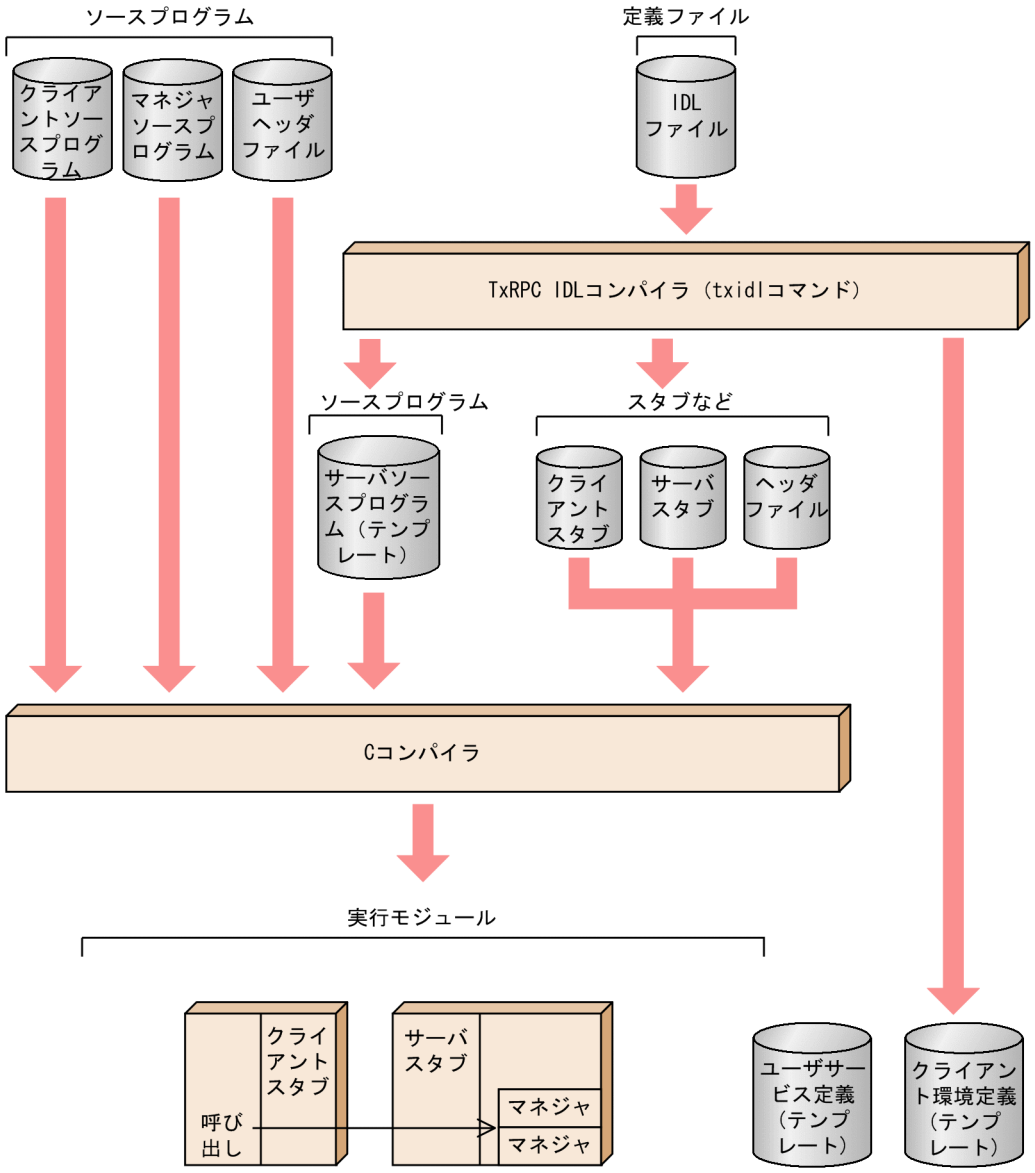
IDL-only TxRPC の通信をする UAP を作成するときの手順は，次のとおりです。

1. **インタフェース定義言語ファイル（IDL ファイル）**を作成します。
2. IDL ファイルを **IDL コンパイラ（txidl コマンド）**でコンパイルします。
3. txidl コマンドで生成されたサーバ UAP のテンプレートを基に，**プログラムをコーディング**します。一緒にクライアント UAP もコーディングします。
4. txidl コマンドで生成されたスタブとコーディングしたプログラムを，**C コンパイラ**でコンパイル，リンケージします。

TxRPC の通信の UAP を作成する手順については，マニュアル「OpenTP1 プログラム作成リファレンス C 言語編」を参照してください。

IDL-only TxRPC で通信する UAP を作成する手順を次の図に示します。

図 6-3　IDL-only TxRPC で通信する UAP を作成する手順



7 TP1/Multi を使う場合の機能

クラスタ／並列システム形態で，TP1/Multi を使う場合の機能について説明します。

7.1　クラスタ／並列システム形態のアプリケーションプログラム

OpenTP1 をクラスタ／並列システムに適用する場合の UAP について説明します。

7.1.1　アプリケーションプログラムを使えるノード

マルチノード機能を使った環境では，被アーカイブジャーナルノードでだけ UAP（ユーザサーバ）を使えます。アーカイブジャーナルノードでは，ユーザサーバを使えません。

7.1.2　アプリケーションプログラムを実行する前提条件

クラスタ／並列システム向けの機能を実行する UAP がある OpenTP1 ノードには，次の前提条件があります。

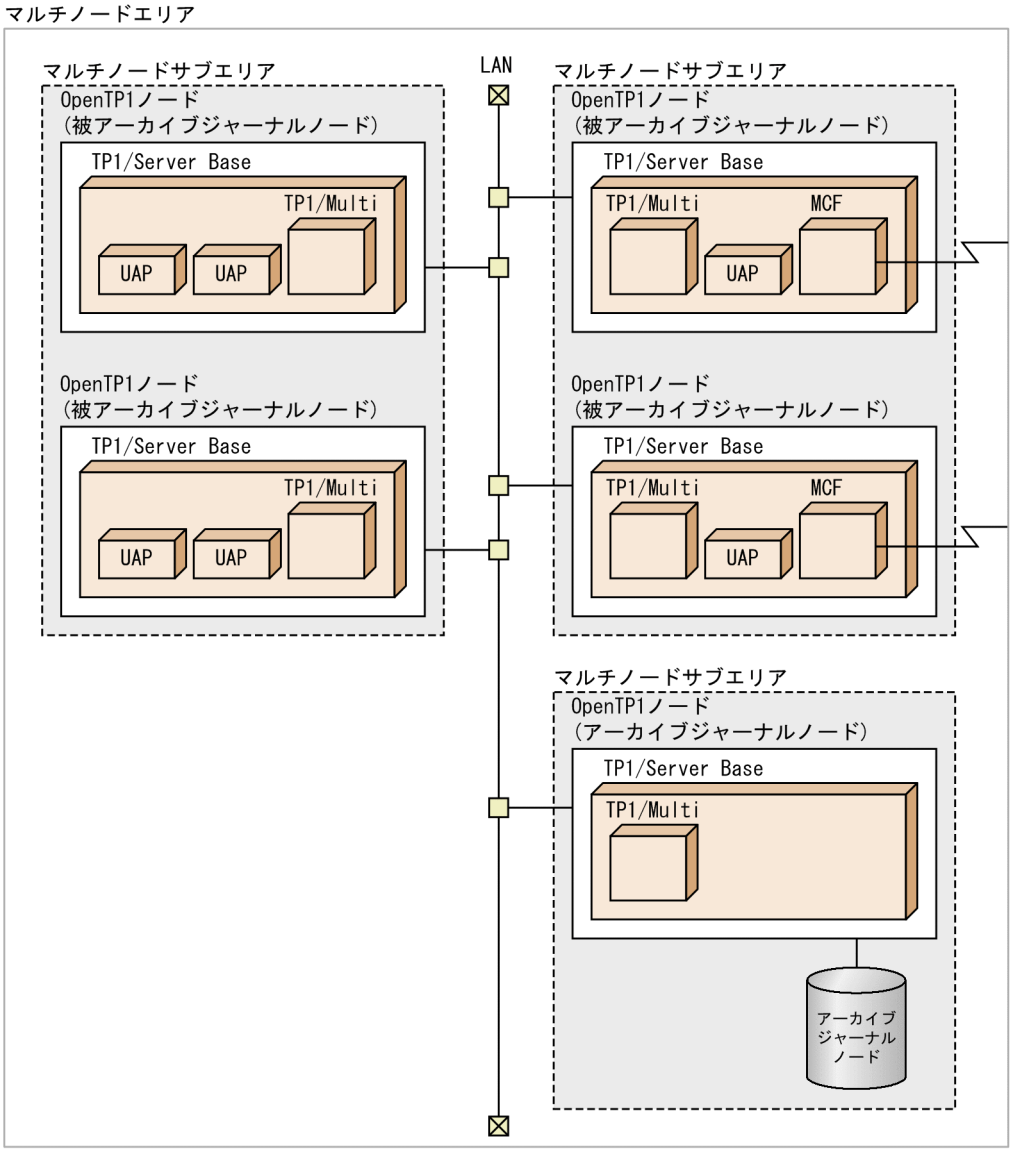
- TP1/Multi が組み込まれている
- システム共通定義の multi_node_option オペランドに Y を指定している

ただし，次に示す機能は，TP1/Multi を組み込んでいなくても，システム共通定義の multi_node_option オペランドに Y を指定していなくてもかまいません。

- ユーザサーバの状態を取得する関数（dc_adm_get_sv_status〜関数）で，自ノードのユーザサーバの状態を取得する場合
- 自 OpenTP1 ノードのノード識別子を取得する関数（dc_adm_get_node_id 関数）を使う場合

クラスタ／並列システム形態のアプリケーションプログラムの概要を次の図に示します。

図 7-1　クラスタ／並列システム形態のアプリケーションプログラムの概要



7.2 アプリケーションプログラムでできる機能

クラスタ／並列システム形態の OpenTP1 の UAP から，OpenTP1 の関数を使ってできる機能について説明します。

7.2.1 OpenTP1 ノードの状態の取得

UAP から関数を呼び出して，クラスタ／並列システムを構成する OpenTP1 ノードの状態を取得できます。OpenTP1 ノードの状態を取得することで，マルチノードエリア，またはマルチノードサブエリアの状態を UAP で監視できます。

取得できる状態を次に示します。

- OpenTP1 ノードが開始していません。
- OpenTP1 ノードが停止中です。または異常終了中です。
- OpenTP1 ノードは正常開始中です。
- OpenTP1 ノードは再開始（リラン）中です。
- OpenTP1 ノードはオンライン中です。
- OpenTP1 ノードは正常終了処理中です。
- OpenTP1 ノードは計画停止 A で終了処理中です。
- OpenTP1 ノードは計画停止 B で終了処理中です。

OpenTP1 ノードの状態を取得するには，ノード状態を連続して取得する方法と，指定した OpenTP1 ノードの状態だけを取得する方法があります。

(1) OpenTP1 ノードの状態を連続して取得する方法

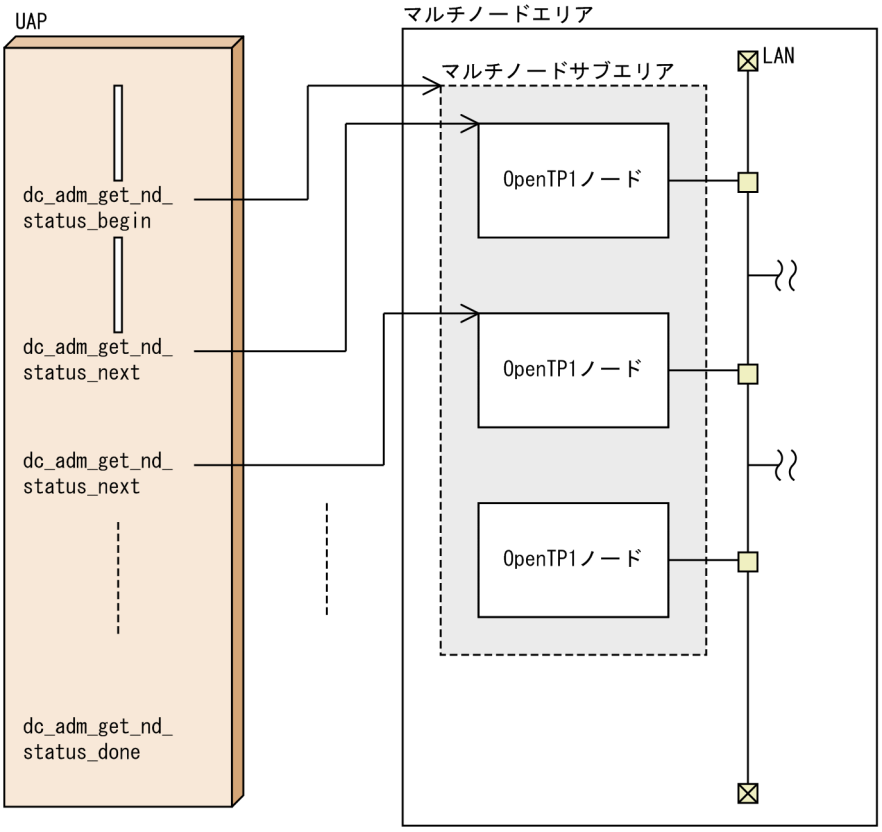
マルチノードエリア，またはマルチノードサブエリア単位に，そのエリアにあるすべての OpenTP1 ノードの状態を取得します。

連続して状態を取得する場合は，取得したいマルチノードエリア，またはマルチノードサブエリアの識別子を指定した dc_adm_get_nd_status_begin 関数を使います。この関数を呼び出すと，指定したエリアにある OpenTP1 ノードの個数がリターンされます。次に，dc_adm_get_nd_status_next 関数を呼び出して，OpenTP1 ノードの状態を取得します。該当する OpenTP1 ノードがなくなるまで dc_adm_get_nd_status_next 関数を呼び出してから，最後に dc_adm_get_nd_status_done 関数を呼び出して状態の取得を終了します。

dc_adm_get_nd_status_begin 関数を呼び出したあとで，もう一度 dc_adm_get_nd_status_begin 関数を呼び出すこと（dc_adm_get_nd_status_begin 関数のネスト）はできません。

OpenTP1 ノードの状態を連続して取得する手順を次の図に示します。

図 7-2　OpenTP1 ノードの状態を連続して取得する手順



(2)　指定した OpenTP1 ノードの状態を取得する方法

一つの OpenTP1 ノードの状態を取得する場合は，dc_adm_get_nd_status 関数を呼び出します。この関数を呼び出すと，関数に設定した OpenTP1 ノードのノード識別子に関する状態を取得できます。

7.2.2　ユーザサーバの状態の取得

UAP から関数を呼び出して，クラスタ／並列システムを構成する OpenTP1 ノードのユーザサーバの状態を取得できます。ユーザサーバの状態を取得することで，UAP でマルチノードエリア，またはマルチノードサブエリアのユーザサーバの状態を監視できます。

取得できる状態を次に示します。

- ユーザサーバが停止中です。または異常終了中です。
- ユーザサーバは正常開始中です。
- ユーザサーバは再開始中です。
- ユーザサーバはオンライン中です。
- ユーザサーバは正常終了処理中です。

ユーザサーバの状態を取得する場合，連続して取得する方法と，指定したユーザサーバの状態だけを取得する方法があります。

(1)　ユーザサーバの状態を連続して取得する方法

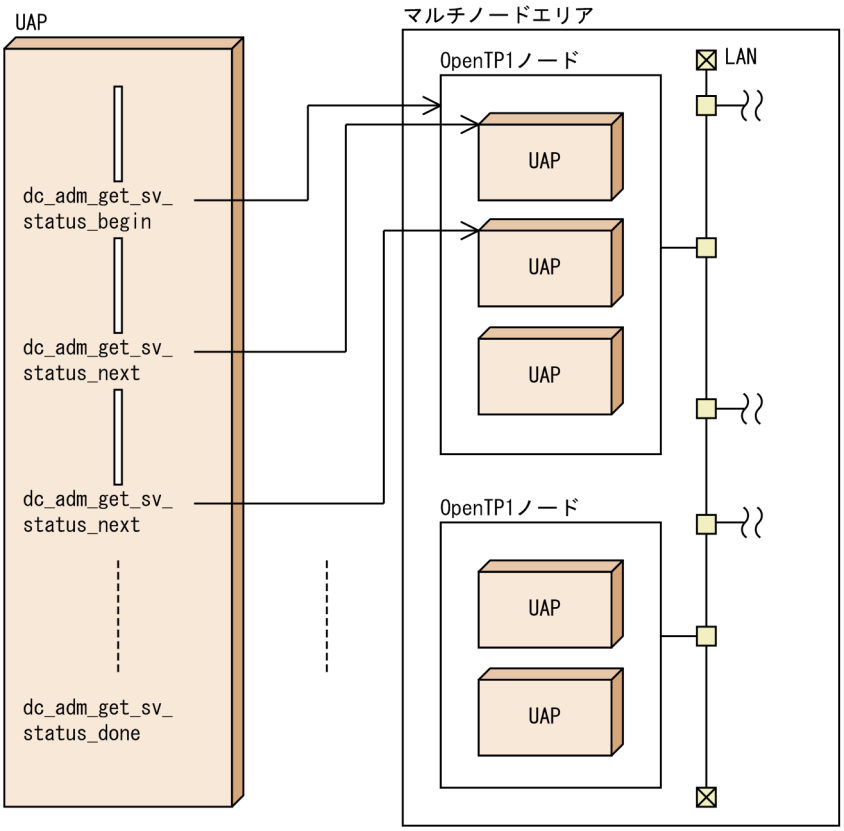
OpenTP1 ノードのノード識別子単位に，そのノードにあるすべてのユーザサーバの状態を取得します。

連続して状態を取得する場合は，取得したいノード識別子を指定した dc_adm_get_sv_status_begin 関数を呼び出します。この関数を呼び出すと，指定した OpenTP1 ノードにあるユーザサーバの個数がリターンされます。次に，dc_adm_get_sv_status_next 関数を呼び出して，ユーザサーバの状態を取得します。該当するユーザサーバがなくなるまで dc_adm_get_sv_status_next 関数を呼び出してから，最後に dc_adm_get_sv_status_done 関数を呼び出して状態の取得を終了します。

dc_adm_get_sv_status_begin 関数を呼び出したあとで，もう一度 dc_adm_get_sv_status_begin 関数を呼び出すこと（dc_adm_get_sv_status_begin 関数のネスト）はできません。

ユーザサーバの状態を連続して取得する手順を次の図に示します。

図 7-3　ユーザサーバの状態を連続して取得する手順



(2)　指定したユーザサーバの状態を取得する方法

一つのユーザサーバの状態を取得する場合は，dc_adm_get_sv_status 関数を呼び出します。この関数を呼び出すと，関数に設定したノード識別子のユーザサーバに関する状態を取得できます。

7.2.3　OpenTP1 ノードのノード識別子の取得

UAP から関数を呼び出して，マルチノードエリア，またはマルチノードサブエリアにあるすべてのノード識別子を取得できます。ノード識別子を取得して，マルチノードエリア，またはマルチノードサブエリアを構成する OpenTP1 ノードを UAP で知ることができます。

OpenTP1 ノードのノード識別子を取得する場合，連続して取得する方法と，指定した OpenTP1 ノードのノード識別子だけを取得する方法があります。

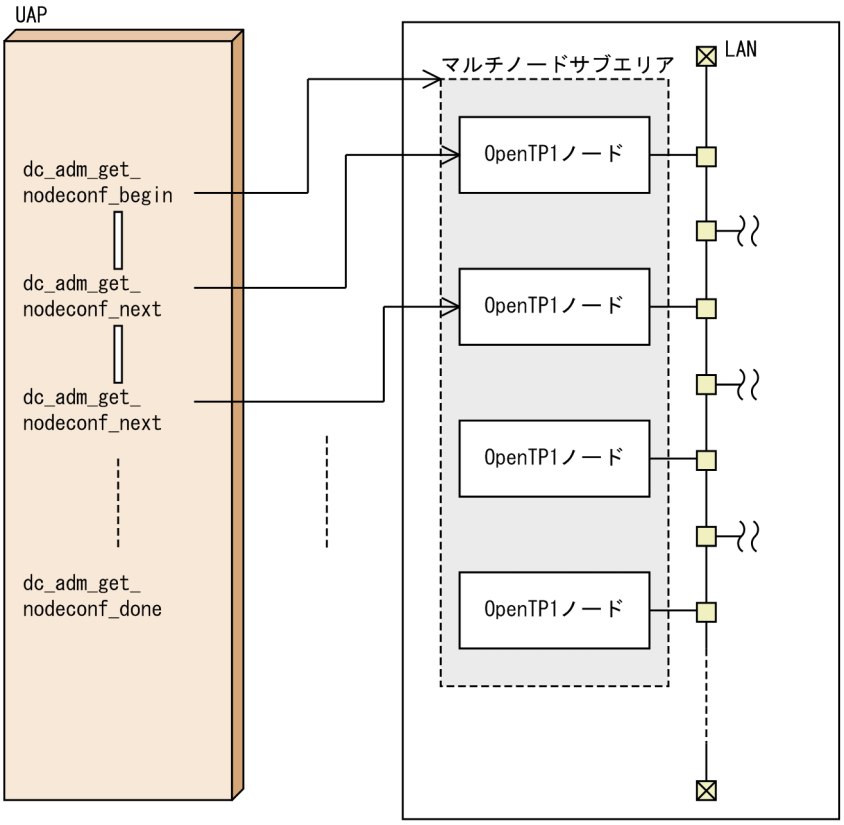
(1)　OpenTP1 ノードのノード識別子を連続して取得する方法

マルチノードエリア，またはマルチノードサブエリア単位に，そのエリアにあるすべての OpenTP1 ノードのノード識別子を取得します。連続してノード識別子を取得する場合は，取得したいエリアを指定した dc_adm_get_nodeconf_begin 関数を呼び出します。この関数を呼び出すと，指定したエリアにある OpenTP1 ノードの個数がリターンされます。次に，dc_adm_get_nodeconf_next 関数を呼び出して，OpenTP1 ノードのノード識別子を取得します。該当する OpenTP1 ノードがなくなるまで dc_adm_get_nodeconf_next 関数を呼び出してから，最後に dc_adm_get_nodeconf_done 関数を呼び出して状態の取得を終了します。

dc_adm_get_nodeconf_begin 関数を呼び出したあとで，もう一度 dc_adm_get_nodeconf_begin 関数を呼び出すこと（dc_adm_get_nodeconf_begin 関数のネスト）はできません。

OpenTP1 ノードのノード識別子を連続して取得する手順を次の図に示します。

図 7-4　OpenTP1 ノードのノード識別子を連続して取得する手順



(2) 自 OpenTP1 ノードのノード識別子を取得する方法

UAP を実行している OpenTP1 ノードのノード識別子を取得する場合は，dc_adm_get_node_id 関数を呼び出します。この関数を呼び出すと，自 OpenTP1 ノードのノード識別子を取得できます。

7.3　マルチノード機能の関数を使える条件

マルチノード機能の関数を使える条件を次の表に示します。

表 7-1　マルチノード機能の関数を使える条件

マルチノードの関数と引数 node_id の指定		TP1/Multi を組み込んでいる		TP1/Multi を組み込んでいない	
		multi_node_option の指定		multi_node_option の指定	
		Y	N	Y	N
dc_adm_get_nd_status_begin	任意	○	×	−	×
dc_adm_get_nd_status_next	任意	○	×	−	×
dc_adm_get_nd_status_done	任意	○	×	−	×
dc_adm_get_nd_status	'*'指定	○	×	−	×
	自 node_id 指定	○	×	−	×
	他 node_id 指定	○	×	−	×
dc_adm_get_sv_status_begin	'*'指定	○	○	−	○
	自 node_id 指定	○	○	−	○
	他 node_id 指定	○	×	−	×
dc_adm_get_sv_status_next	'*'指定	○	○	−	○
	自 node_id 指定	○	○	−	○
	他 node_id 指定	○	×	−	×
dc_adm_get_sv_status_done	'*'指定	○	○	−	○
	自 node_id 指定	○	○	−	○
	他 node_id 指定	○	×	−	×
dc_adm_get_sv_status	'*'指定	○	○	−	○
	自 node_id 指定	○	○	−	○
	他 node_id 指定	○	×	−	×
dc_adm_get_nodeconf_begin	任意	○	×	−	×
dc_adm_get_nodeconf_next	任意	○	×	−	×
dc_adm_get_nodeconf_done	任意	○	×	−	×
dc_adm_get_node_id	任意	○	○	−	○

（凡例）

○：該当する条件で使えます。

×：該当する条件では使えません。

－：該当する条件で関数を呼び出すと，OpenTP1 が異常終了します。

8 OpenTP1 のサンプル

OpenTP1 で提供するサンプルの使い方について説明します。

8.1　サンプルの概要

システムをより手軽に構築するため，OpenTP1 では**サンプル（例題）**を使えます。サンプルを使うと，次のような利点があります。

- OpenTP1 を組み込んでから確立するまでの負担を減らせます。
- システムの運用を支援するためのツールが使えます。
- UAP を COBOL 言語でコーディングするときに，テンプレートが使えます。

8.1.1　サンプルの種類

OpenTP1 のサンプルを次に示します。

- Base サンプル
  TP1/Server Base と一緒に提供するサンプルです。
- DAM サンプル
  TP1/FS/Direct Access と一緒に提供するサンプルです。
- TAM サンプル
  TP1/FS/Table Access と一緒に提供するサンプルです。
- MCF サンプル
  メッセージ制御機能（TP1/Message Control，TP1/NET/Library，および通信プロトコル対応製品）と一緒に提供するサンプルです。
- delvcmd コマンド
  マルチ OpenTP1 に対するコマンドを振り分けます。
- COBOL 言語用テンプレート
  UAP を COBOL 言語でコーディングするときに使える，データ部のテンプレートです。
- サンプルシナリオテンプレート
  シナリオテンプレートを利用したシステムの運用時に使用するサンプルシナリオテンプレートです。このサンプルを使用するには，シナリオテンプレートを利用したシステムの前提となる JP1 製品（JP1/AJS，JP1/AJS2 - Scenario Operation，JP1/Base）が必要です。
- リアルタイム取得項目定義テンプレート
  リアルタイム統計情報サービスで使用するテンプレートです。

それぞれのサンプルは独立していて，ディレクトリごとに分けて格納してあります。サンプルを使うときは，それぞれ独立して使えます。

(1) アプリケーションプログラムからアクセスするファイル

Base サンプル，DAM サンプル，TAM サンプルは，サンプルプログラム（UAP）と OpenTP1 ファイルシステムを使えます。UAP は，同じ形態のプログラムを使っています。ただし，UAP で使うデータベースを格納する場所は，サンプルによって異なります。

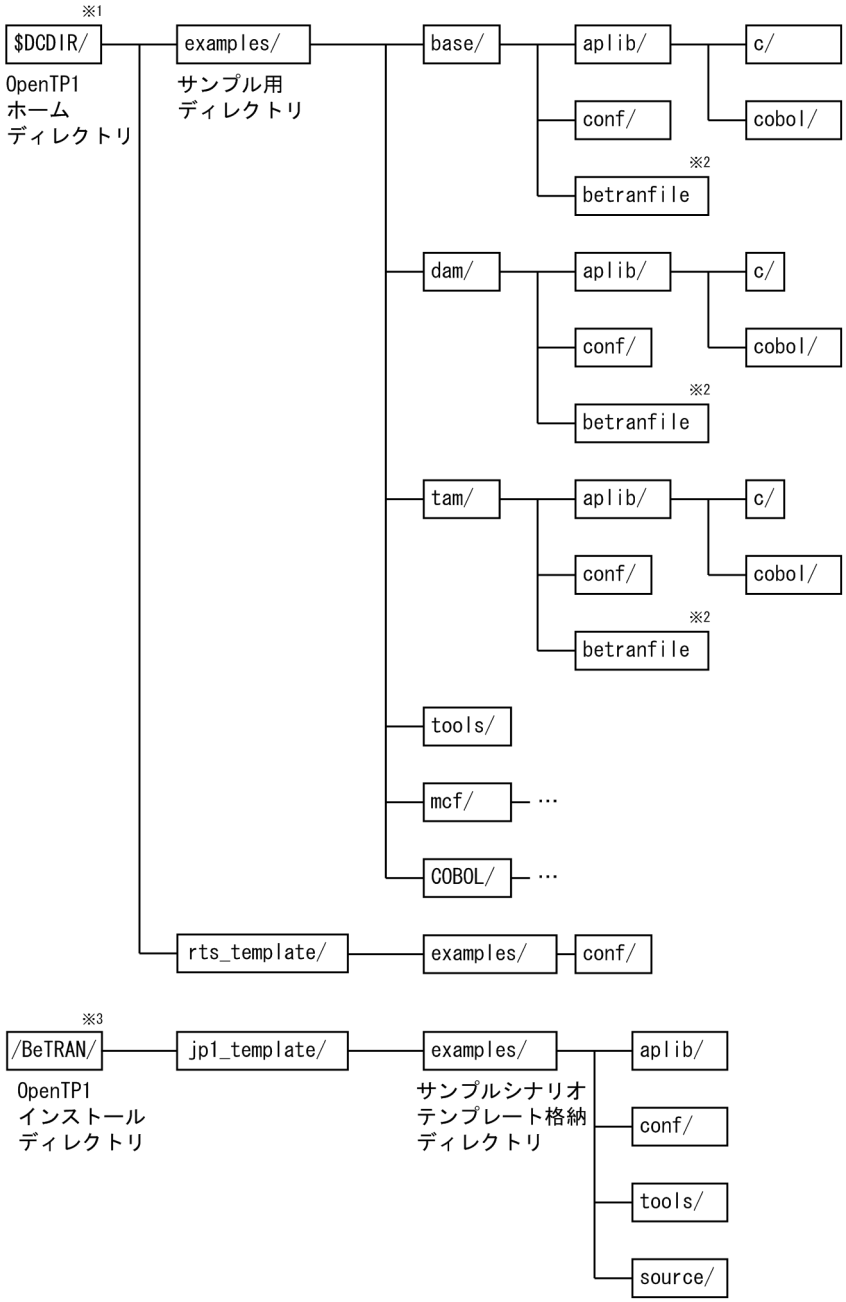
- Base サンプルのデータベース：**メモリテーブル上**
- DAM サンプルのデータベース：**DAM ファイル**
- TAM サンプルのデータベース：**TAM テーブル**

上記のサンプルの UAP では，データベースに対して，データの参照や更新などのアクセスをします。このアクセス手順で，OpenTP1 の API を使う方法がわかります。

8.1.2 サンプルのディレクトリ構成

OpenTP1 のサンプルで使うファイルについて説明します。OpenTP1 のサンプルが格納されているディレクトリを次の図に示します。

図 8-1　サンプルを格納するディレクトリ構成



注※1

$DCDIR は，OpenTP1 ホームディレクトリを示す環境変数です。OpenTP1 のサンプル一式をデフォルトの OpenTP1 ホームディレクトリ以外のディレクトリにコピーした場合は，そのディレクトリ名を示します。OpenTP1 のサンプル一式をコピーしていない場合は，examples/ディレクトリは$DCDIR の下にあります。

注※2

betranfile は，ツール用のサンプルのコマンドを実行すると作成されます。初期状態では作成されていません。

注※3

OpenTP1 のインストールディレクトリは OS によって異なります。

(1) $DCDIR 直下の examples/ディレクトリの内容

$DCDIR 直下の examples/ディレクトリには，サンプルで使うディレクトリが格納してあります。examples/ディレクトリの各ファイルの内容を次に示します。

base/

Base サンプルのファイルが格納してあるディレクトリ

dam/

DAM サンプルのファイルが格納してあるディレクトリ

tam/

TAM サンプルのファイルが格納してあるディレクトリ

tools/

サンプルで共通に使うツールが格納してあるディレクトリ（ツールディレクトリ）

mcf/

メッセージ制御機能（MCF）サンプルのファイルが格納してあるディレクトリ

COBOL/

COBOL 言語用テンプレートを格納するディレクトリ

(a) base/，dam/，tam/ディレクトリの内容

base/，dam/，tam/ディレクトリに格納してあるディレクトリ，およびファイルを次に示します。

aplib/

サンプルの UAP が格納されているディレクトリ

c/

各サンプルの UAP のソースファイル（C 言語）が格納されているディレクトリ

cobol/

各サンプルの UAP のソースファイル（COBOL 言語）が格納されているディレクトリ

conf/

各サンプル用の定義ファイルが格納されているディレクトリ

betranfile

各サンプル用の OpenTP1 ファイルシステム（このファイルの内容は tools/ディレクトリのツールを使って作成します）

(b) tools/ディレクトリの内容

tools/ディレクトリの各ファイルの内容を次に示します。

base_mkfs

Base サンプル用の OpenTP1 ファイルシステムを作成するシェルファイル

dam_mkfs

DAM サンプル用の OpenTP1 ファイルシステムを作成するシェルファイル

apbat

DAM サンプル用で DAM ファイルを作成するファイル（dam_mkfs で使います）。このファイルは，make コマンドを実行して UAP の実行形式ファイルを作成したときに作成されます。実行形式ファイルの作成方法については，「8.3.2(1)(b)　UAP の実行形式ファイルの作成」を参照してください。

tam_mkfs

TAM サンプル用の OpenTP1 ファイルシステムを作成するシェルファイル

tamdata

TAM テーブル作成用のデータファイル（tam_mkfs で使います）

chconf

定義ファイルを修正するコマンド（$DCDIR を実際の OpenTP1 ホームディレクトリに変更するときに使います）

bkconf

chconf で変更した定義ファイルを元に戻すコマンド

delvcmd

マルチ OpenTP1 形態のノードにコマンドを振り分けるコマンド。delvcmd コマンドについては，「8.7　マルチ OpenTP1 のコマンドを振り分けるサンプル」を参照してください。

(c)　mcf/ディレクトリの内容

mcf/ディレクトリの構成については，「8.6　MCF サンプルの使い方」を参照してください。

(d)　COBOL/ディレクトリの内容

COBOL/ディレクトリの構成については，「8.8　COBOL 言語用テンプレート」を参照してください。

(2)　$DCDIR 直下の rts_template/ディレクトリの内容

$DCDIR 直下の rts_template/ディレクトリには，リアルタイム統計情報サービスで使うディレクトリが格納してあります。rts_template/ディレクトリの各ファイルの内容を次に示します。

(a)　examples/ディレクトリの内容

examples/ディレクトリの各ファイルの内容を次に示します。

conf/

リアルタイム統計情報サービス用のリアルタイム取得項目定義ファイルが格納されているディレクトリ

- base_itm

BASE 用のリアルタイム取得項目定義ファイル

- dam_itm
  DAM 用のリアルタイム取得項目定義ファイル
- tam_itm
  TAM 用のリアルタイム取得項目定義ファイル
- all_itm
  すべての統計情報を取得するリアルタイム取得項目定義ファイル
- none_itm
  すべての統計情報を取得しないリアルタイム取得項目定義ファイル
- mcfs_itm
  MCF 用のリアルタイム取得項目定義ファイル（システム全体，サーバ，サービス単位に取得する項目）
- mcfl_itm
  MCF 用のリアルタイム取得項目定義ファイル（論理端末単位に取得する項目）
- mcfg_itm
  MCF 用のリアルタイム取得項目定義ファイル（サービスグループ単位に取得する項目）

(3) インストールディレクトリ直下の jp1_template/ディレクトリの内容

インストールディレクトリ直下の jp1_template/ディレクトリには，サンプルシナリオテンプレートで使うディレクトリが格納してあります。jp1_template/ディレクトリの各ファイルの内容を次に示します。

examples/
　スケールアウトのサンプルシナリオテンプレートで使用するファイルが格納されているディレクトリ

(a) examples/ディレクトリの内容

examples/ディレクトリの各ディレクトリの内容を次に示します。

aplib/
　シナリオテンプレートのサンプルプログラムのロードモジュール（source/ディレクトリ下のソースファイルに対するロードモジュール）が格納されているディレクトリ

conf/
　サンプルシナリオテンプレート用の定義ファイルが格納されているディレクトリ

tools/
　サンプルシナリオテンプレートで使用するシェルファイルが格納されているディレクトリ

- dcjset_conf
  サンプルシナリオテンプレート用の OpenTP1 の環境設定をするシェルファイル

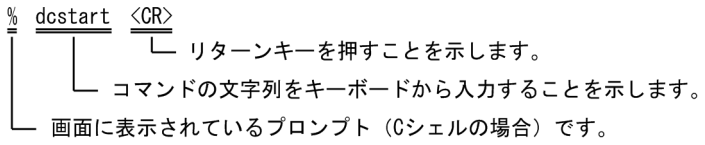
- dcj_mkfs
  サンプルシナリオテンプレート用の OpenTP1 のファイルシステムを作成するシェルファイル
- dcjmk_dcdir
  サンプルシナリオテンプレート用の OpenTP1 のディレクトリを作成するシェルファイル

source/

シナリオテンプレートのサンプルプログラム（UAP）が格納されているディレクトリ。サンプルシナリオテンプレートでは，Base サンプルをサンプルプログラムとして使用しています。Base サンプルの仕様については，「8.5　サンプルプログラムの仕様」を参照してください。

8.1.3　サンプルの説明方法

サンプルの説明で使う，コマンド入力例を表記する形式について説明します。コマンド入力例は，次のように表記します。



ユーザが入力するコマンドは，アンダーライン（_）で示します。上の例は「画面にプロンプトが表示されているのを確認した上で，dcstart コマンドを入力してリターンキーを押す」ことを示します。

8.2　Base サンプルの使い方

Base サンプルを使う手順について説明します。サンプルを使う手順の概要を次の図に示します。

図 8-2　サンプルを使う手順の概要（Base サンプル：スタブを使用する場合）

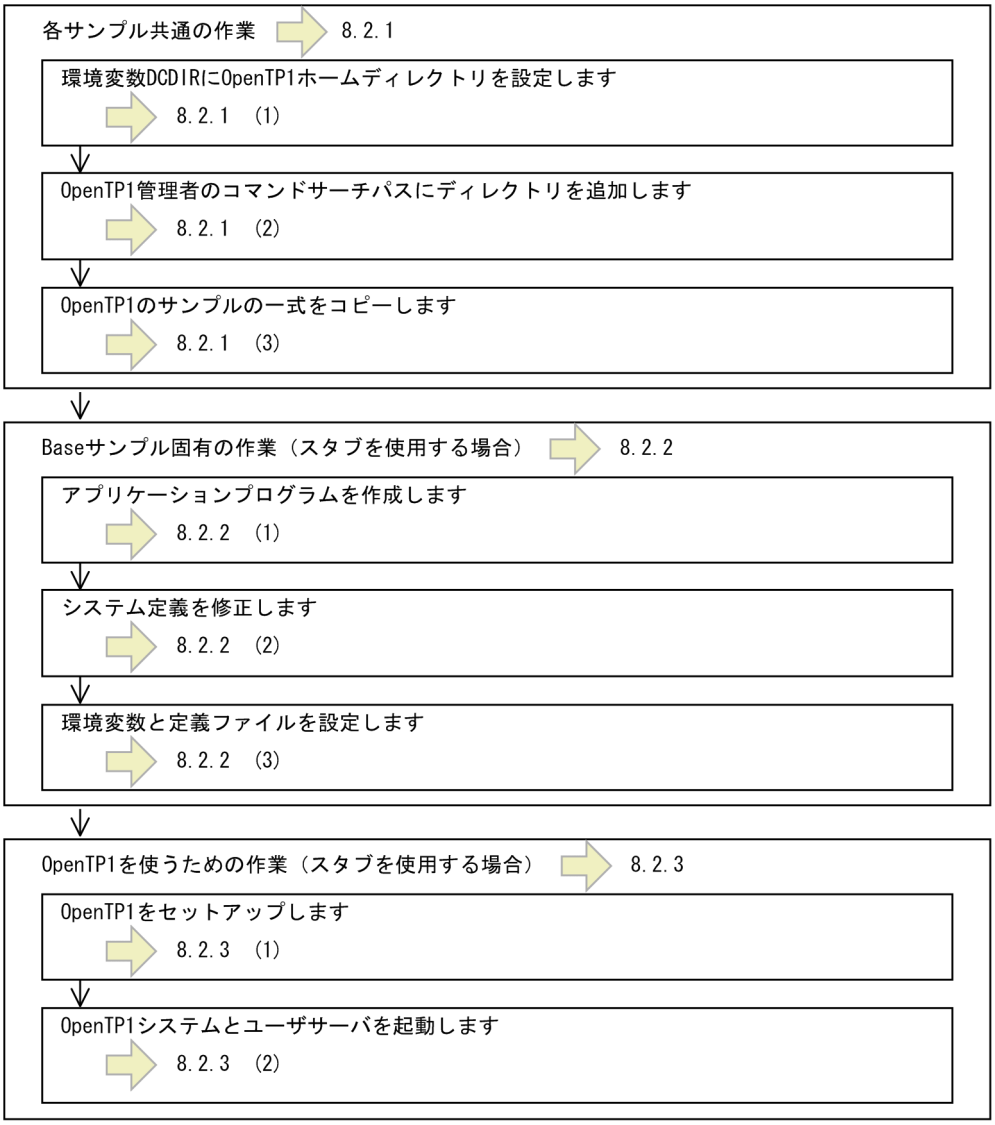
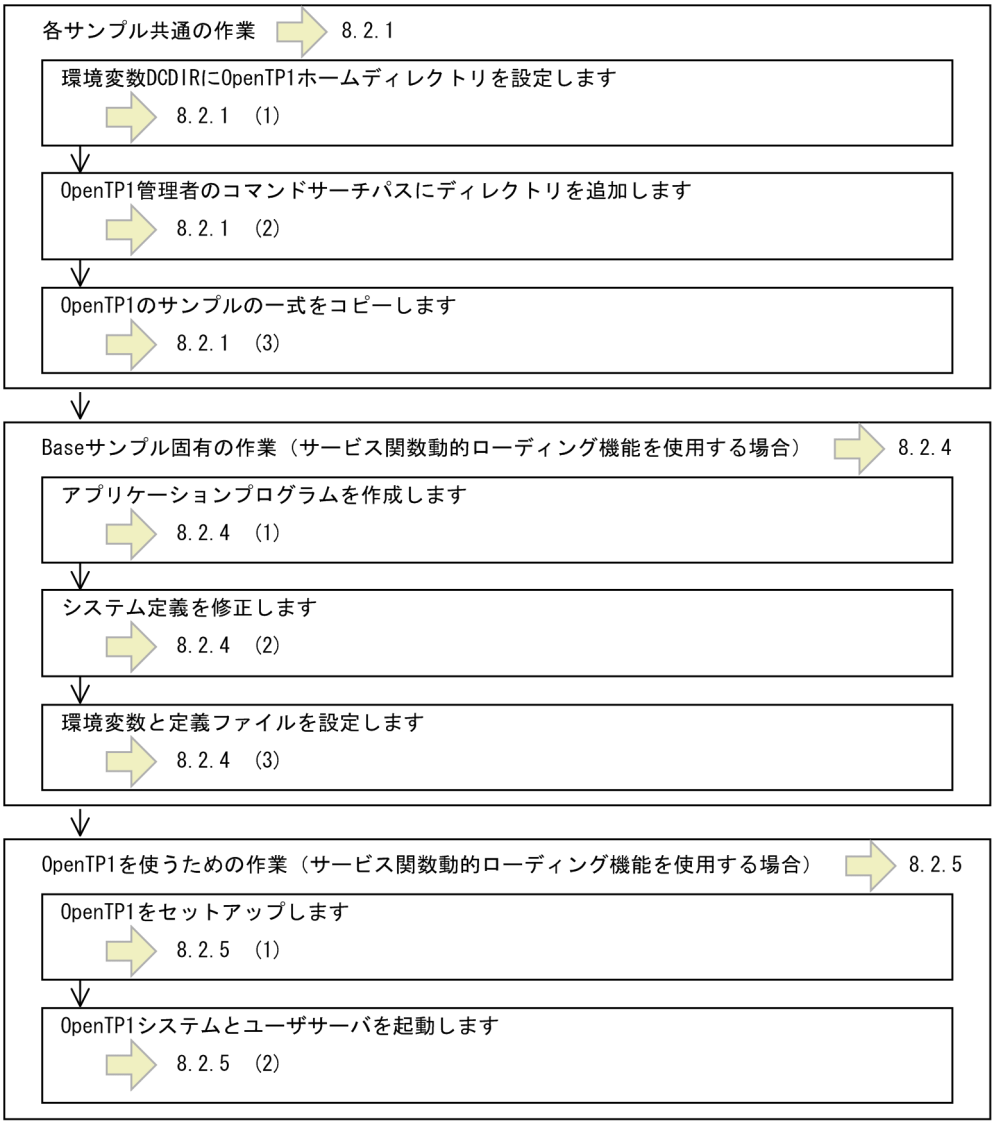


図 8-3　サンプルを使う手順の概要（Base サンプル：サービス関数動的ローディング機能を使用する場合）



8.2.1　サンプル共通の作業（Base サンプル）

OpenTP1 の 3 種類のサンプル（Base サンプル，DAM サンプル，TAM サンプル）に共通する準備手順について説明します。

サンプルを使う前には，OpenTP1 の製品（TP1/Server Base，TP1/FS/Direct Access，TP1/FS/Table Access）を組み込んでおきます。この作業は，OpenTP1 管理者が操作します。ここまでは，通常の OpenTP1 の組み込み手順と同じです。

(1)　環境変数 DCDIR に OpenTP1 ホームディレクトリを設定します

環境変数 DCDIR に OpenTP1 ホームディレクトリを設定します。

（例）OpenTP1 ホームディレクトリを/usr/betran にする場合

csh のとき

```
% setenv DCDIR /usr/betran <CR>
```

bash のとき

```
$ export DCDIR=/usr/betran <CR>
```

(2) OpenTP1 管理者のコマンドサーチパスにディレクトリを追加します

OpenTP1 管理者のコマンドのサーチパスに，OpenTP1 のコマンドのサーチパスと，サンプルで使うコマンドのサーチパスを追加します。

- $DCDIR/bin：OpenTP1 のコマンドのサーチパスです。
- $DCDIR/examples/tools：サンプル用のツールのサーチパスです。

(3) OpenTP1 のサンプル一式をコピーします

OpenTP1 ホームディレクトリを/BeTRAN 以外に変更した場合は，OpenTP1 ホームディレクトリの環境に，サンプル一式をコピーします。OpenTP1 ホームディレクトリを/BeTRAN のままでサンプルを使う場合は，コピーする必要はありません。コピーするときは，UNIX の cp，tar などのコマンドを使います。

コピーするときは，OpenTP1 ホームディレクトリの下に，「8.1.2　サンプルのディレクトリ構成」で示す構成になるようにしてください。これ以外の構成とした場合，システムの動作は保証しません。

（例）

OpenTP1 を組み込んだディレクトリが/BeTRAN で，そのディレクトリから OpenTP1 ホームディレクトリにサンプルをコピーする場合（cp コマンドを使います）

```
% cp -R /BeTRAN/examples $DCDIR <CR>
```

これで，OpenTP1 のサンプルの環境設定は終了です。続けて，各サンプル固有の作業を実行します。

8.2.2　Base サンプル固有の作業（スタブを使用する場合）

Base サンプル固有の準備手順について説明します。ここの記述では，次の条件でサンプルを使うものとします。

使うシェル：C シェル

OpenTP1 ホームディレクトリ：/usr/betran

(1)　アプリケーションプログラムを作成します

Base サンプルの UAP を作成する手順を示します。サンプルプログラムを作成するときは，UNIX のツール **make コマンド**を使います。サンプルでは，専用の **makefile** を Base サンプルのディレクトリに作成してあります。

make コマンドは，aplib/ディレクトリにある c/または cobol/ディレクトリをカレントディレクトリとして実行します。make コマンドは，OpenTP1 をセットアップしたあとに実行してください。OpenTP1 のセットアップについては，「8.2.3(1)　OpenTP1 をセットアップします」を参照してください。
OpenTP1 をセットアップする前に make コマンドを実行するとコンパイルエラーで失敗します。

C 言語の UAP を作成するコマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/base/aplib/c  <CR>
%  make   <CR>
```

このコマンドを実行すると，aplib/ディレクトリの下に basespp および basesup という UAP の実行形式ファイル（C 言語）が作成されます。

(2)　システム定義を修正します

サンプルでは，システム定義の手間を省くために，定義ファイルの例を提供しています。ただし，一部の定義ファイルは，実際の OpenTP1 ホームディレクトリを絶対パス名で指定する必要があります。

(a)　OpenTP1 ホームディレクトリの定義を変更する手順

実際の OpenTP1 ホームディレクトリに修正するときは，変更用のツール **chconf コマンド**を使います。このコマンドを実行すると，定義ファイルの項目で OpenTP1 ホームディレクトリが$DCDIR となっている部分を，実際の OpenTP1 ホームディレクトリ（例えば，/usr/betran）に変更できます。

chconf コマンドは，$DCDIR/examples/base/conf/ディレクトリに移動してから実行します。コマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/base/conf  <CR>
%  chconf  <CR>
```

chconf コマンドを実行すると，定義ファイルの内容が変更されます。変更する定義ファイルと変更する内容を次の表に示します。実際の OpenTP1 ホームディレクトリに変更される部分を太字で示します。

表 8-1　変更する定義ファイルと変更する内容（Base サンプル）

変更する定義ファイル	変更する内容
env	putenv DCCONFPATH **$DCDIR**/examples/base/conf
prc	putenv prcsvpath **$DCDIR**/examples/base/aplib
sts	物理ファイル名：**$DCDIR**/examples/base/betranfile/×××

変更する定義ファイル	変更する内容
sysjnl	物理ファイル名：$DCDIR/examples/base/betranfile/×××
cdtrn	物理ファイル名：$DCDIR/examples/base/betranfile/×××

このツール（chconf コマンド）を実行する前には，環境変数 DCDIR にあらかじめ OpenTP1 ホームディレクトリを設定しておいてください。設定していないと正常に変更できません。

(b) 変更した OpenTP1 ホームディレクトリを元に戻す方法

変更した OpenTP1 ホームディレクトリを元に戻すときは，変更取り消し用のツール **bkconf コマンド**を使います。chconf コマンドで変更しようとした内容を，初期状態に戻します。

chconf コマンドでシステム定義を正常に変更できなかった場合は，すぐに bkconf コマンドを実行してください。

コマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/base/conf  <CR>
%  bkconf  <CR>
```

(3) 環境変数と定義ファイルを設定します

作成したサンプル UAP とサンプルのシステム定義で，OpenTP1 システムを開始する手順について説明します。

(a) 環境変数 DCCONFPATH の設定

環境変数 DCCONFPATH に，定義ファイルを格納しているディレクトリを設定します。この設定をすると，OpenTP1 が定義ファイルの内容を認識できるようになります。

コマンド入力例を次に示します。

csh の場合

```
%  setenv  DCCONFPATH  $DCDIR/examples/base/conf  <CR>
```

bash の場合

```
$  export  DCCONFPATH=$DCDIR/examples/base/conf  <CR>
```

(b) 定義ファイル env のコピー

定義ファイルのうち，**env ファイル**だけは，$DCDIR/conf から OpenTP1 に読み込まれます。そのため，サンプルとして作成した env 定義ファイルを$DCDIR/conf へ移動します。

$DCDIR/conf/ディレクトリに env 定義ファイルを作成している場合は，上書きされてしまうので，必要であれば退避しておいてください。

コマンド入力例を次に示します。

```
% cp $DCDIR/examples/base/conf/env $DCDIR/conf <CR>
```

(c) OpenTP1 ファイルシステムの初期化

OpenTP1 ファイルシステムを初期化します。Base サンプル用の OpenTP1 ファイルシステムの初期化は，base_mkfs というシェルファイルを実行します。

コマンド入力例を次に示します。

```
% base_mkfs <CR>
```

このシェルファイルを実行すると，$DCDIR/examples/base/ディレクトリの下に betranfile というファイルが生成され，この下に OpenTP1 ファイルシステムが構築されます。

8.2.3 OpenTP1 を使うための作業（スタブを使用する場合）

システム定義を修正し終わったら，OpenTP1 を使うための作業をします。

(1) OpenTP1 をセットアップします

OpenTP1 をセットアップするときは，dcsetup コマンドを実行します。dcsetup コマンドは，/BeTRAN/bin/ディレクトリの下にあります。

コマンド入力例を次に示します。

```
% /BeTRAN/bin/dcsetup OpenTP1ホームディレクトリ名 <CR>
```

セットアップの作業は，スーパユーザが操作します。dcsetup コマンドは，絶対パス名で実行してください。dcsetup コマンドについては，マニュアル「OpenTP1 運用と操作」を参照してください。

サンプルプログラムを作成する make コマンドは，OpenTP1 をセットアップしたあとに実行してください。OpenTP1 をセットアップする前に make コマンドを実行するとコンパイルエラーで失敗します。

(2) OpenTP1 システムとユーザサーバを起動します

OpenTP1 システムとユーザサーバを起動する手順について説明します。

(a) OpenTP1 システムの起動

OpenTP1 システムを dcstart コマンドで起動します。

コマンド入力例を次に示します。

```
% dcstart <CR>
```

(b) ユーザサーバ（UAP）の起動

dcsvstart コマンドで，作成した UAP を起動します。サーバ UAP（SPP）を起動してから，クライアント UAP（SUP）を起動します。

コマンド入力例を次に示します。

```
% dcsvstart -u basespp <CR>
  basesppがオンライン状態になったことがメッセージログで出力されます。
% dcsvstart -u basesup <CR>
  basesupがオンライン状態になったことがメッセージログで出力されます。
  ユーザサーバ（UAP）の処理経過が出力されます。
```

サーバ UAP（SPP）は，ユーザサービス構成定義で OpenTP1 システムの起動時に自動的に起動することもできます。

(3) OpenTP1 ファイルシステムの内容一覧

OpenTP1 ファイルシステム作成ツール base_mkfs コマンドを実行すると，$DCDIR/examples/base/betranfile ファイル上に OpenTP1 ファイルシステムが作成されます。作成される OpenTP1 ファイルシステムの内容を次の表に示します。

表 8-2 OpenTP1 ファイルシステムの内容一覧（Base サンプル）

ファイル名	使う目的となるファイル	レコード長※	レコード数
jnl01	システムジャーナルファイル	4096 バイト	50 レコード
jnl02	システムジャーナルファイル	4096 バイト	50 レコード
jnl03	システムジャーナルファイル	4096 バイト	50 レコード
stsfil01	ステータスファイル	4608 バイト	50 レコード
stsfil02	ステータスファイル	4608 バイト	50 レコード
stsfil03	ステータスファイル	4608 バイト	50 レコード
stsfil04	ステータスファイル	4608 バイト	50 レコード
cpdf01	チェックポイントダンプファイル	4096 バイト	256 レコード
cpdf02	チェックポイントダンプファイル	4096 バイト	256 レコード
cpdf03	チェックポイントダンプファイル	4096 バイト	256 レコード

注※

ここで示すレコード長は，省略時仮定値です。

(4)　サンプル UAP の入れ替え

サンプルの UAP は，次に示す手順で入れ替えてください。

1. OpenTP1 システムを停止します。
2. dcsetup コマンドに-d オプションを付けて実行して，いったん OpenTP1 を OS から削除します。
3. 「8.2　Base サンプルの使い方」で示す手順で，使いたいサンプルの UAP を設定し直します。
4. UAP を実行します。

8.2.4　Base サンプル固有の作業（サービス関数動的ローディング機能を使用する場合）

サービス関数動的ローディング機能を使用する場合の，Base サンプル固有の準備手順について説明します。ここの記述では，次の条件でサンプルを使うものとします。

使うシェル：C シェル

OpenTP1 ホームディレクトリ：/usr/betran

(1)　アプリケーションプログラムを作成します

サービス関数動的ローディング機能を使用する場合の，Base サンプルの UAP を作成する手順を示します。サンプルプログラムを作成するときは，UNIX のツール make コマンドを使います。サンプルでは，専用の makefile を Base サンプルのディレクトリに作成してあります。

make コマンドは，aplib/ディレクトリにある c/または cobol/ディレクトリをカレントディレクトリとして実行します。make コマンドは，OpenTP1 をセットアップしたあとに実行してください。OpenTP1 のセットアップについては，「8.2.5(1)　OpenTP1 をセットアップします」を参照してください。OpenTP1 をセットアップする前に make コマンドを実行するとコンパイルエラーで失敗します。

C 言語の UAP を作成するコマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/base/aplib/c  <CR>
%  make   -f make svdl  <CR>
```

このコマンドを実行すると，aplib/ディレクトリの下に basespp2 および basesup2 という UAP の実行形式ファイル（C 言語）が作成されます。

(2)　システム定義を修正します

サンプルでは，システム定義の手間を省くために，定義ファイルの例を提供しています。ただし，一部の定義ファイルは，実際の OpenTP1 ホームディレクトリを絶対パス名で指定する必要があります。

(a) OpenTP1 ホームディレクトリの定義を変更する手順

実際の OpenTP1 ホームディレクトリに修正するときは，変更用のツール chconf コマンドを使います。このコマンドを実行すると，定義ファイルの項目で OpenTP1 ホームディレクトリが$DCDIR となっている部分を，実際の OpenTP1 ホームディレクトリ（例えば，/usr/betran）に変更できます。

chconf コマンドは，$DCDIR/examples/base/conf/ディレクトリに移動してから実行します。コマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/base/conf  <CR>
%  chconf  <CR>
```

chconf コマンドを実行すると，定義ファイルの内容が変更されます。変更する定義ファイルと変更する内容を次の表に示します。実際の OpenTP1 ホームディレクトリに変更される部分を太字で示します。

表 8-3　変更する定義ファイルと変更する内容（Base サンプル）

変更する定義ファイル	変更する内容
env	putenv DCCONFPATH **$DCDIR**/examples/base/conf
prc	putenv prcsvpath **$DCDIR**/examples/base/aplib
sts	物理ファイル名：**$DCDIR**/examples/base/betranfile/×××
sysjnl	物理ファイル名：**$DCDIR**/examples/base/betranfile/×××
cdtrn	物理ファイル名：**$DCDIR**/examples/base/betranfile/×××

このツール（chconf コマンド）を実行する前には，環境変数 DCDIR にあらかじめ OpenTP1 ホームディレクトリを設定しておいてください。設定していないと正常に変更できません。

なお，basespp2 および BASESPP2 の service オペランドに指定された，$DCDIR を含む UAP 共用ライブラリ名は，環境変数を使用して指定できるため，chconf コマンドを実行しても定義ファイルの内容は変更されません。

(b) 変更した OpenTP1 ホームディレクトリを元に戻す方法

変更した OpenTP1 ホームディレクトリを元に戻すときは，変更取り消し用のツール bkconf コマンドを使います。chconf コマンドで変更しようとした内容を，初期状態に戻します。

chconf コマンドでシステム定義を正常に変更できなかった場合は，すぐに bkconf コマンドを実行してください。

コマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/base/conf  <CR>
%  bkconf  <CR>
```

(3) 環境変数と定義ファイルを設定します

作成したサンプル UAP とサンプルのシステム定義で，OpenTP1 システムを開始する手順について説明します。

(a) 環境変数 DCCONFPATH の設定

環境変数 DCCONFPATH に，定義ファイルを格納しているディレクトリを設定します。この設定をすると，OpenTP1 が定義ファイルの内容を認識できるようになります。

コマンド入力例を次に示します。

csh の場合

```
% setenv DCCONFPATH $DCDIR/examples/base/conf <CR>
```

bash の場合

```
$ export DCCONFPATH=$DCDIR/examples/base/conf <CR>
```

(b) 定義ファイル env のコピー

定義ファイルのうち，env ファイルだけは，$DCDIR/conf から OpenTP1 に読み込まれます。そのため，サンプルとして作成した env 定義ファイルを$DCDIR/conf へ移動します。

$DCDIR/conf/ディレクトリに env 定義ファイルを作成している場合は，上書きされてしまうので，必要であれば退避しておいてください。

コマンド入力例を次に示します。

```
% cp $DCDIR/examples/base/conf/env $DCDIR/conf <CR>
```

(c) OpenTP1 ファイルシステムの初期化

OpenTP1 ファイルシステムを初期化します。Base サンプル用の OpenTP1 ファイルシステムの初期化は，base_mkfs というシェルファイルを実行します。

コマンド入力例を次に示します。

```
% base_mkfs <CR>
```

このシェルファイルを実行すると，$DCDIR/examples/base/ディレクトリの下に betranfile というファイルが生成され，この下に OpenTP1 ファイルシステムが構築されます。

8.2.5 OpenTP1 を使うための作業（サービス関数動的ローディング機能を使用する場合）

システム定義を修正し終わったら，OpenTP1 を使うための作業をします。

(1) OpenTP1 をセットアップします

OpenTP1 をセットアップするときは，dcsetup コマンドを実行します。dcsetup コマンドは，/BeTRAN/bin/ディレクトリの下にあります。

コマンド入力例を次に示します。

```
% /BeTRAN/bin/dcsetup OpenTP1ホームディレクトリ名 <CR>
```

セットアップの作業は，スーパユーザが操作します。dcsetup コマンドは，絶対パス名で実行してください。dcsetup コマンドについては，マニュアル「OpenTP1 運用と操作」を参照してください。

サンプルプログラムを作成する make コマンドは，OpenTP1 をセットアップしたあとに実行してください。OpenTP1 をセットアップする前に make コマンドを実行するとコンパイルエラーで失敗します。

(2) OpenTP1 システムとユーザサーバを起動します

OpenTP1 システムとユーザサーバを起動する手順について説明します。

(a) OpenTP1 システムの起動

OpenTP1 システムを dcstart コマンドで起動します。

コマンド入力例を次に示します。

```
% dcstart <CR>
```

(b) ユーザサーバ（UAP）の起動

dcsvstart コマンドで，作成した UAP を起動します。サーバ UAP（SPP）を起動してから，クライアント UAP（SUP）を起動します。

コマンド入力例を次に示します。

```
% dcsvstart -u basespp2 <CR>
  basespp2がオンライン状態になったことがメッセージログで出力されます。
% dcsvstart -u basesup2 <CR>
  basesup2がオンライン状態になったことがメッセージログで出力されます。
```

ユーザサーバ（UAP）の処理経過が出力されます。

サーバ UAP（SPP）は，ユーザサービス構成定義で OpenTP1 システムの起動時に自動的に起動することもできます。

(3) OpenTP1 ファイルシステムの内容一覧

OpenTP1 ファイルシステム作成ツール base_mkfs コマンドを実行すると，$DCDIR/examples/base/betranfile ファイル上に OpenTP1 ファイルシステムが作成されます。作成される OpenTP1 ファイルシステムの内容を次の表に示します。

表 8-4 OpenTP1 ファイルシステムの内容一覧（Base サンプル）

ファイル名	使う目的となるファイル	レコード長※	レコード数
jnl01	システムジャーナルファイル	4096 バイト	50 レコード
jnl02	システムジャーナルファイル	4096 バイト	50 レコード
jnl03	システムジャーナルファイル	4096 バイト	50 レコード
stsfil01	ステータスファイル	4608 バイト	50 レコード
stsfil02	ステータスファイル	4608 バイト	50 レコード
stsfil03	ステータスファイル	4608 バイト	50 レコード
stsfil04	ステータスファイル	4608 バイト	50 レコード
cpdf01	チェックポイントダンプファイル	4096 バイト	256 レコード
cpdf02	チェックポイントダンプファイル	4096 バイト	256 レコード
cpdf03	チェックポイントダンプファイル	4096 バイト	256 レコード

注※

ここで示すレコード長は，省略時仮定値です。

(4) サンプル UAP の入れ替え

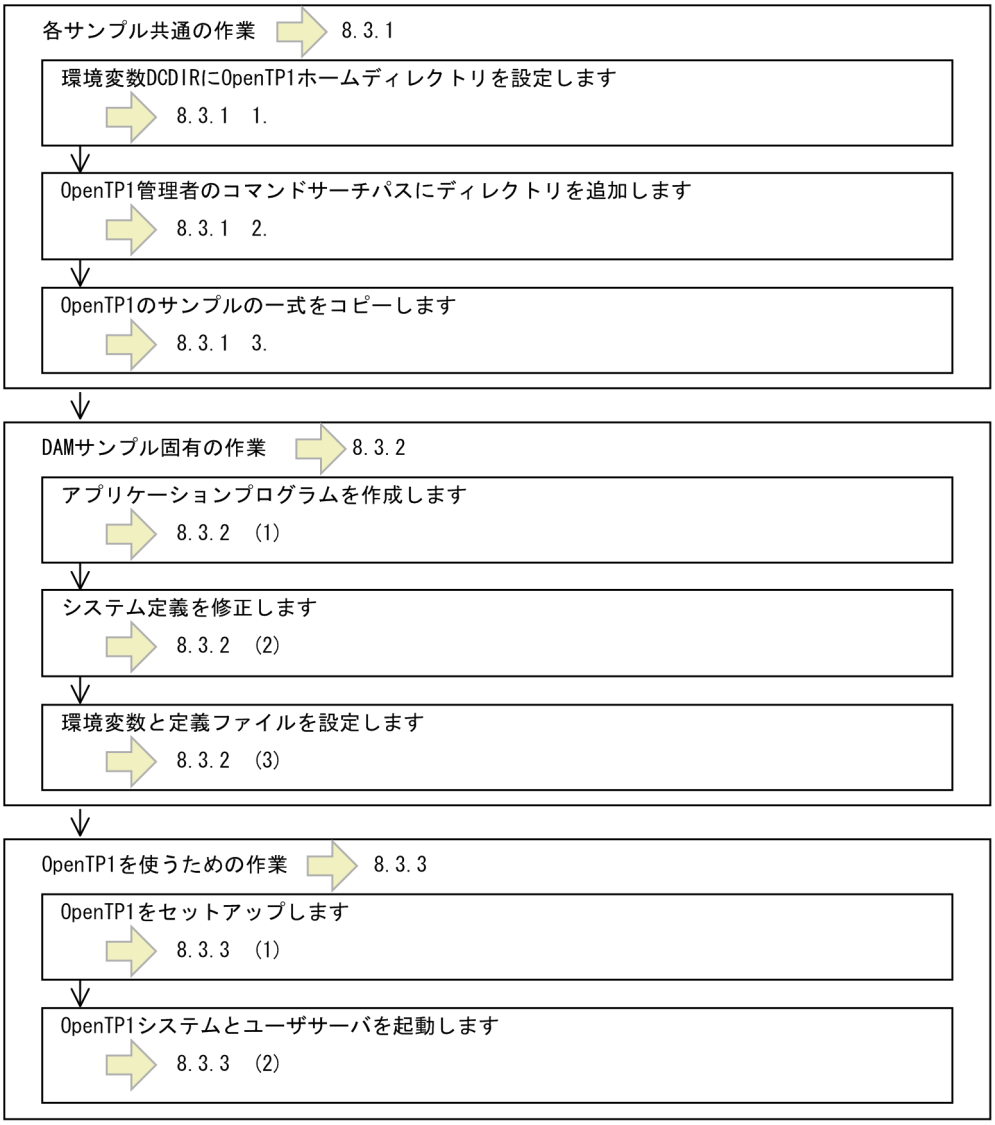
サンプルの UAP は，次に示す手順で入れ替えてください。

1. OpenTP1 システムを停止します。
2. dcsetup コマンドに-d オプションを付けて実行して，いったん OpenTP1 を OS から削除します。
3. 「8.2 Base サンプルの使い方」で示す手順で，使いたいサンプルの UAP を設定し直します。
4. UAP を実行します。

8.3　DAM サンプルの使い方

DAM サンプルを使う手順について説明します。サンプルを使う手順の概要を次の図に示します。

図 8-4　サンプルを使う手順の概要（DAM サンプル）



8.3.1　サンプル共通の作業（DAM サンプル）

次に示す手順は，OpenTP1 の 3 種類のサンプルに共通する準備です。

1. 環境変数 DCDIR に OpenTP1 ホームディレクトリを設定します
2. OpenTP1 管理者のコマンドサーチパスにディレクトリを追加します
3. OpenTP1 のサンプル一式をコピーします

上記の手順については，「8.2.1　サンプル共通の作業（Base サンプル）」を参照してください。

8.3.2　DAM サンプル固有の作業

DAM サンプル固有の準備手順について説明します。ここの記述では，次の条件でサンプルを使うものとします。

使うシェル：C シェル

OpenTP1 ホームディレクトリ：/usr/betran

(1)　アプリケーションプログラムを作成します

DAM サンプル用のサンプルプログラムを作成する手順を示します。サンプルプログラムは，UNIX のツール **make コマンド**を使います。サンプルでは，専用の **makefile** を DAM サンプルのディレクトリに作成してあります。make コマンドは，OpenTP1 をセットアップしたあとに実行してください。OpenTP1 のセットアップについては，「8.3.3(1)　OpenTP1 をセットアップします」を参照してください。
OpenTP1 をセットアップする前に make コマンドを実行するとコンパイルエラーで失敗します。

(a)　トランザクション制御用オブジェクトファイルの作成

OpenTP1 のコマンド **trnmkobj コマンド**で，**トランザクション制御用オブジェクトファイル**を作成します。

サンプルで使う makefile 内では，トランザクション制御用オブジェクトファイルを **dam_sw.o** という名称で作成するようにしてあります。そのため，trnmkobj コマンドを実行して作成するオブジェクトファイル名は，dam_sw.o になるようにしてください。

オブジェクトファイルは，**$DCDIR/spool/trnrmcmd/userobj/ディレクトリ**の下に作成されます。同じ名称のオブジェクトファイルがある場合は，事前に退避させておいてください。

コマンド入力例を次に示します。

```
%  trnmkobj  -o  dam_sw  -R  OpenTP1_DAM  <CR>
```

(b)　UAP の実行形式ファイルの作成

aplib/ディレクトリにある c/または cobol/ディレクトリをカレントディレクトリとして，**make コマンド**を実行します。

C 言語の UAP を作成するコマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/dam/aplib/c  <CR>
%  make   <CR>
```

このコマンドを実行すると，aplib/ディレクトリに UAP の実行形式ファイル（C 言語）が作成されます。

(2) システム定義を修正します

サンプルでは，システム定義の手間を省くために，システム定義の定義例を提供しています。ただし，一部の定義ファイルは，実際の OpenTP1 ホームディレクトリを絶対パス名で指定する必要があります。

(a) OpenTP1 ホームディレクトリの定義を変更する手順

実際の OpenTP1 ホームディレクトリに修正するときは，変更用のツール chconf コマンドを使います。このコマンドを実行と，定義ファイルに OpenTP1 ホームディレクトリが$DCDIR となっている部分を OpenTP1 ホームディレクトリ（例えば，/usr/betran）に変更します。

chconf コマンドは，conf/ディレクトリに移動してから実行します。

コマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/dam/conf  <CR>
%  chconf  <CR>
```

chconf コマンドを実行すると，定義ファイルの内容が変更されます。変更する定義ファイルと変更する内容を次の表に示します。変更される部分を太字で示します。

表 8-5　変更する定義ファイルと変更する内容（DAM サンプル）

変更する定義ファイル	変更する内容
env	putenv DCCONFPATH **$DCDIR**/examples/dam/conf
prc	putenv prcsvpath **$DCDIR**/examples/dam/aplib
sts	物理ファイル名：**$DCDIR**/examples/dam/betranfile/×××
sysjnl	物理ファイル名：**$DCDIR**/examples/dam/betranfile/×××
cdtrn	物理ファイル名：**$DCDIR**/examples/dam/betranfile/×××

このツール（chconf コマンド）を実行する前には，環境変数 DCDIR にあらかじめ OpenTP1 ホームディレクトリを設定しておいてください。設定していないと正常に変更できません。

(b) 変更した OpenTP1 ホームディレクトリを元に戻す方法

変更した OpenTP1 ホームディレクトリを元に戻すときは，変更取り消し用のツール bkconf コマンドを使います。chconf コマンドで変更しようとした内容を，初期状態に戻します。

chconf コマンドでシステム定義を正常に変更できなかった場合は，すぐに bkconf コマンドを実行してください。

コマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/dam/conf  <CR>
%  bkconf  <CR>
```

(3) 環境変数と定義ファイルを設定します

作成したサンプル UAP とサンプルのシステム定義で，OpenTP1 システムを開始する手順について説明します。

(a) 環境変数 DCCONFPATH の設定

環境変数 DCCONFPATH に定義ファイルを格納しているディレクトリを設定します。この設定をすると，OpenTP1 が定義ファイルの内容を認識できるようになります。

コマンド入力例を次に示します。

csh の場合

```
% setenv DCCONFPATH $DCDIR/examples/dam/conf <CR>
```

bash の場合

```
$ export DCCONFPATH=$DCDIR/examples/dam/conf <CR>
```

(b) 定義ファイル env のコピー

定義ファイルのうち，env ファイルだけは，$DCDIR/conf から OpenTP1 に読み込まれます。そのため，サンプルとして作成した env 定義ファイルを$DCDIR/conf へ移動します。

$DCDIR/conf/ディレクトリに env 定義ファイルを作成している場合は，上書きされてしまうので，必要であれば退避しておいてください。

コマンド入力例を次に示します。

```
% cp $DCDIR/examples/dam/conf/env $DCDIR/conf <CR>
```

(c) OpenTP1 ファイルシステムの初期化

OpenTP1 ファイルシステムを初期化します。DAM サンプル用の OpenTP1 ファイルシステムの初期化は，dam_mkfs というシェルファイルを実行します。

コマンド入力例を次に示します。

```
% dam_mkfs <CR>
```

このシェルファイルを実行すると，$DCDIR/examples/dam/ディレクトリの下に betranfile というファイルが生成され，この下に OpenTP1 ファイルシステムが構築されます。

8.3.3 OpenTP1 を使うための作業

システム定義を修正し終わったら，OpenTP1 を使うための作業をします。

(1) OpenTP1 をセットアップします

OpenTP1 をセットアップするときは，dcsetup コマンドを実行します。dcsetup コマンドは，/BeTRAN/bin/ディレクトリの下にあります。

コマンド入力例を次に示します。

```
% /BeTRAN/bin/dcsetup OpenTP1ホームディレクトリ名 <CR>
```

セットアップの作業は，スーパユーザが操作します。dcsetup コマンドは，絶対パス名で実行してください。dcsetup コマンドについては，マニュアル「OpenTP1 運用と操作」を参照してください。

サンプルプログラムを作成する make コマンドは，OpenTP1 をセットアップしたあとに実行してください。OpenTP1 をセットアップする前に make コマンドを実行するとコンパイルエラーで失敗します。

(2) OpenTP1 システムとユーザサーバを起動します

作成したサンプル UAP とサンプルのシステム定義で，OpenTP1 システムを開始する手順について説明します。

(a) OpenTP1 システムの起動

OpenTP1 システムを dcstart コマンドで起動します。

コマンド入力例を次に示します。

```
% dcstart <CR>
```

(b) ユーザサーバ（UAP）の起動

dcsvstart コマンドで，作成した UAP を起動します。サーバ UAP（SPP）を起動してから，クライアント UAP（SUP）を起動します。

コマンド入力例を次に示します。

```
% dcsvstart -u damspp <CR>
  damsppがオンライン状態になったことがメッセージログで出力されます。
% dcsvstart -u damsup <CR>
  damsupがオンライン状態になったことがメッセージログで出力されます。
  ユーザサーバ（UAP）の処理経過がメッセージログに出力されます。
```

サーバ UAP（SPP）は，ユーザサービス構成定義で OpenTP1 システムの起動時に自動的に起動することもできます。

(3) OpenTP1 ファイルシステムの内容一覧

OpenTP1 ファイルシステム作成ツール dam_mkfs を実行すると，$DCDIR/examples/dam/betranfile/ ディレクトリの下に OpenTP1 ファイルシステムが作成されます。作成される OpenTP1 ファイルシステムの内容を次の表に示します。

表 8-6 OpenTP1 ファイルシステムの内容一覧（DAM サンプル）

ファイル名	使う目的となるファイル	レコード長	レコード数
jnlf01	システムジャーナルファイル	4096 バイト	100 レコード
jnlf02	システムジャーナルファイル	4096 バイト	100 レコード
jnlf03	システムジャーナルファイル	4096 バイト	100 レコード
stsfil01	ステータスファイル	4608 バイト	64 レコード
stsfil02	ステータスファイル	4608 バイト	64 レコード
stsfil03	ステータスファイル	4608 バイト	64 レコード
stsfil04	ステータスファイル	4608 バイト	64 レコード
cpdf01	チェックポイントダンプファイル	4096 バイト	100 レコード
cpdf02	チェックポイントダンプファイル	4096 バイト	100 レコード
cpdf03	チェックポイントダンプファイル	4096 バイト	100 レコード
smplfile	DAM ファイル	512 バイト※	11 ブロック

注※

DAM ファイルの欄は，ブロック長を示します。

(4) サンプル UAP の入れ替え

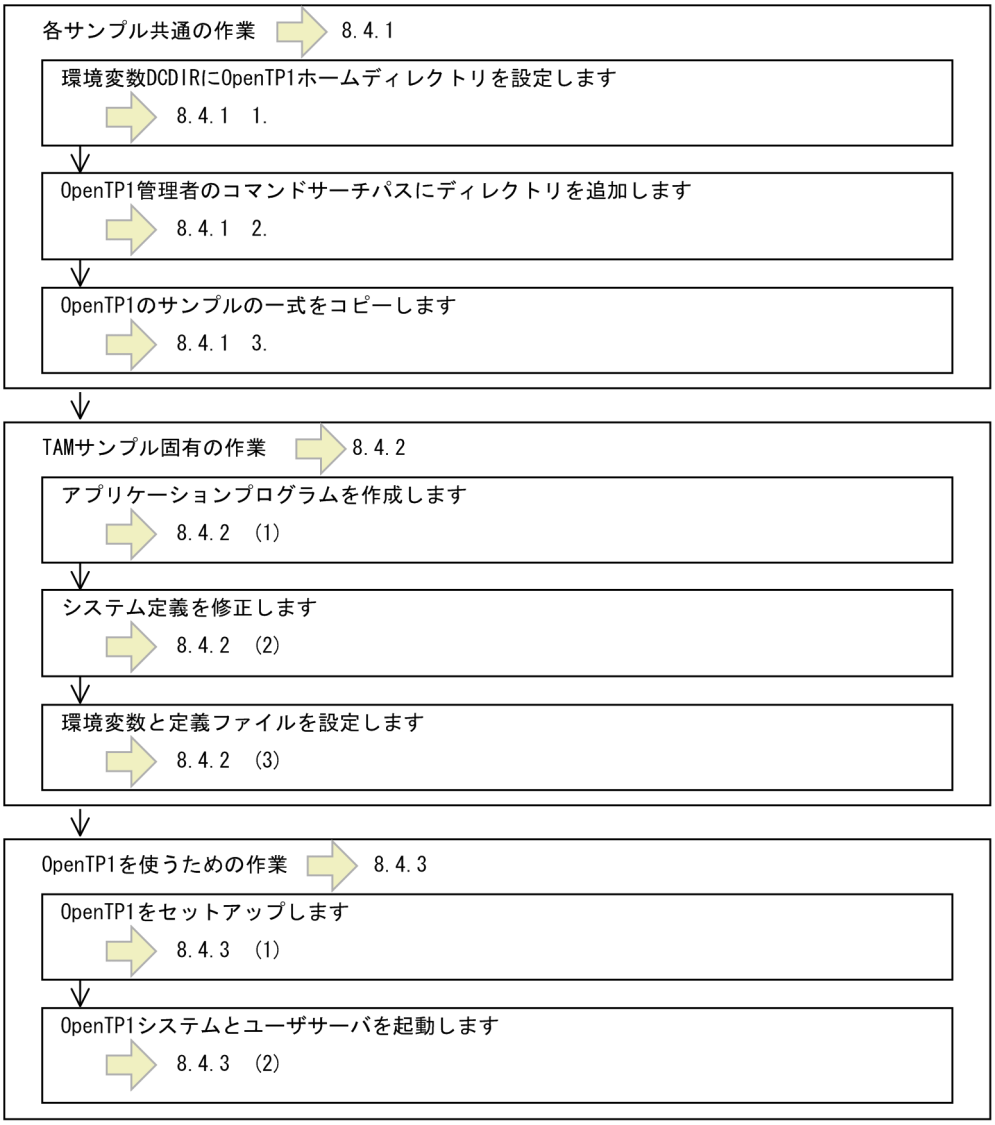
サンプルの UAP は，次に示す手順で入れ替えてください。

1. OpenTP1 システムを停止します。
2. dcsetup コマンドに-d オプションを付けて実行して，いったん OpenTP1 を OS から削除します。
3. 「8.3 DAM サンプルの使い方」で示す手順で，使いたいサンプルの UAP を設定し直します。
4. UAP を実行します。

8.4　TAM サンプルの使い方

TAM サンプルを使う手順について説明します。サンプルを使う手順の概要を次の図に示します。

図 8-5　サンプルを使う手順の概要（TAM サンプル）



8.4.1　サンプル共通の作業（TAM サンプル）

次に示す手順は，OpenTP1 の 3 種類のサンプルに共通する準備です。

1. 環境変数 DCDIR に OpenTP1 ホームディレクトリを設定します
2. OpenTP1 管理者のコマンドサーチパスにディレクトリを追加します
3. OpenTP1 のサンプル一式をコピーします

上記の手順については，「8.2.1　サンプル共通の作業（Base サンプル）」を参照してください。

8.4.2　TAM サンプル固有の作業

TAM サンプル固有の準備手順について説明します。ここの記述では，次の条件でサンプルを使うものとします。

使うシェル：C シェル

OpenTP1 ホームディレクトリ：/usr/betran

(1)　アプリケーションプログラムを作成します

TAM サンプル用のサンプルプログラムを作成する手順を示します。サンプルプログラムは，UNIX のツール **make コマンド**を使います。サンプルでは，専用の **makefile** を TAM サンプルのディレクトリに作成してあります。make コマンドは，OpenTP1 をセットアップしたあとに実行してください。OpenTP1 のセットアップについては，「8.4.3(1)　OpenTP1 をセットアップします」を参照してください。
OpenTP1 をセットアップする前に make コマンドを実行するとコンパイルエラーで失敗します。

(a)　トランザクション制御用オブジェクトファイルの作成

OpenTP1 のコマンド **trnmkobj コマンド**で，**トランザクション制御用オブジェクトファイル**を作成します。

サンプルで使う makefile 内では，トランザクション制御用オブジェクトファイルを **tam_sw.o** という名称で作成するようにしてあります。そのため，trnmkobj コマンドを実行して作成するオブジェクトファイル名は，tam_sw.o になるようにしてください。

オブジェクトファイルは，**$DCDIR/spool/trnrmcmd/userobj/ディレクトリ**の下に作成されます。同じ名称のオブジェクトファイルがある場合は，事前に退避させておいてください。

コマンド入力例を次に示します。

```
%  trnmkobj  -o  tam_sw  -R  OpenTP1_TAM  <CR>
```

(b)　UAP の実行形式ファイルの作成

aplib/ディレクトリにある c/または cobol/ディレクトリをカレントディレクトリとして，**make コマンド**を実行します。C 言語の UAP を作成するコマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/tam/aplib/c  <CR>
%  make  <CR>
```

このコマンドを実行すると，aplib/ディレクトリに UAP の実行形式ファイル（C 言語）が作成されます。

(2)　システム定義を修正します

サンプルでは，システム定義の手間を省くために，システム定義の初期値を提供しています。ただし，一部の定義ファイルは，実際の OpenTP1 ホームディレクトリを絶対パス名で指定する必要があります。

(a) OpenTP1 ホームディレクトリの定義を変更する手順

実際の OpenTP1 ホームディレクトリに修正するときは，変更用のツール chconf コマンドを使います。このコマンドを実行することで，定義ファイルに OpenTP1 ホームディレクトリが$DCDIR となっている部分を OpenTP1 ホームディレクトリ（例えば，/usr/betran）に変更します。

chconf コマンドは，サンプルで提供する conf/ディレクトリに移動してから実行します。コマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/tam/conf  <CR>
%  chconf  <CR>
```

chconf コマンドを実行すると，定義ファイルの内容が変更されます。変更する定義ファイルと変更する内容を次の表に示します。変更される部分を太字で示します。

表 8-7　変更する定義ファイルと変更する内容（TAM サンプル）

変更する定義ファイル	変更する内容
env	putenv DCCONFPATH **$DCDIR**/examples/tam/conf
prc	putenv prcsvpath **$DCDIR**/examples/tam/aplib
sts	物理ファイル名：**$DCDIR**/examples/tam/betranfile/×××
sysjnl	物理ファイル名：**$DCDIR**/examples/tam/betranfile/×××
cdtrn	物理ファイル名：**$DCDIR**/examples/tam/betranfile/×××

このツール（chconf コマンド）を実行する前には，環境変数 DCDIR にあらかじめ OpenTP1 ホームディレクトリを設定しておいてください。設定していないと正常に変更できません。

(b) 変更した OpenTP1 ホームディレクトリを元に戻す方法

変更した OpenTP1 ホームディレクトリを元に戻すときは，変更取り消し用のツール bkconf コマンドを使います。chconf コマンドで変更しようとした内容を，初期状態に戻します。

chconf コマンドでシステム定義を正常に変更できなかった場合は，すぐに bkconf コマンドを実行してください。

コマンド入力例を次に示します。

```
%  chdir  $DCDIR/examples/tam/conf  <CR>
%  bkconf  <CR>
```

(3) 環境変数と定義ファイルを設定します

作成したサンプル UAP とサンプルのシステム定義で，OpenTP1 システムを開始する手順について説明します。

(a) 環境変数 DCCONFPATH の設定

環境変数 DCCONFPATH に定義ファイルを格納しているディレクトリを設定します。この設定をすることで，OpenTP1 が定義ファイルの内容を認識できるようになります。

コマンド入力例を次に示します。

csh の場合

```
% setenv DCCONFPATH $DCDIR/examples/tam/conf <CR>
```

bash の場合

```
$ export DCCONFPATH=$DCDIR/examples/tam/conf <CR>
```

(b) 定義ファイル env のコピー

定義ファイルのうち，env ファイルだけは，$DCDIR/conf から OpenTP1 に読み込まれます。そのため，サンプルとして作成した env 定義ファイルを$DCDIR/conf へ移動します。

$DCDIR/conf/ディレクトリに env 定義ファイルを作成している場合は，上書きされてしまうので，必要であれば退避しておいてください。

コマンド入力例を次に示します。

```
% cp $DCDIR/examples/tam/conf/env $DCDIR/conf <CR>
```

(c) OpenTP1 ファイルシステムの初期化

OpenTP1 ファイルシステムを初期化します。TAM サンプル用の OpenTP1 ファイルシステムの初期化は，tam_mkfs というシェルファイルを実行します。

コマンド入力例を次に示します。

```
% tam_mkfs <CR>
```

このシェルファイルを実行することで，$DCDIR/examples/tam/ディレクトリの下に betranfile というファイルが生成され，この下に OpenTP1 ファイルシステムが構築されます。

8.4.3 OpenTP1 を使うための作業

システム定義を修正し終わったら，OpenTP1 を使うための作業をします。

(1) OpenTP1 をセットアップします

OpenTP1 をセットアップするときは，dcsetup コマンドを実行します。dcsetup コマンドは，/BeTRAN/bin/ディレクトリの下にあります。

コマンド入力例を次に示します。

```
% /BeTRAN/bin/dcsetup OpenTP1ホームディレクトリ名 <CR>
```

セットアップの作業は，スーパユーザが操作します。dcsetup コマンドは，絶対パス名で実行してください。dcsetup コマンドについては，マニュアル「OpenTP1 運用と操作」を参照してください。

サンプルプログラムを作成する make コマンドは，OpenTP1 をセットアップしたあとに実行してください。OpenTP1 をセットアップする前に make コマンドを実行するとコンパイルエラーで失敗します。

(2) OpenTP1 システムとユーザサーバを起動します

作成したサンプル UAP とサンプルのシステム定義で，OpenTP1 システムを開始する手順について説明します。

(a) OpenTP1 システムの起動

OpenTP1 システムを dcstart コマンドで起動します。コマンド入力例を次に示します。

```
% dcstart <CR>
```

(b) ユーザサーバ（UAP）の起動

dcsvstart コマンドで，作成した UAP を起動します。サーバ UAP（SPP）を起動してから，クライアント UAP（SUP）を起動します。コマンド入力例を次に示します。

```
% dcsvstart -u tamspp <CR>
  tamsppがオンライン状態になったことがメッセージログで出力されます。
% dcsvstart -u tamsup <CR>
  tamsupがオンライン状態になったことがメッセージログで出力されます。
  ユーザサーバ（UAP）の処理経過が出力されます。
```

サーバ UAP（SPP）は，ユーザサービス構成定義で OpenTP1 システムの起動時に自動的に起動することもできます。

(3) OpenTP1 ファイルシステムの内容一覧

OpenTP1 ファイルシステム作成ツール tam_mkfs を実行すると，$DCDIR/examples/tam/betranfile/ ディレクトリの下に OpenTP1 ファイルシステムが作成されます。

作成される OpenTP1 ファイルシステムの内容を，次の表に示します。

表 8-8 OpenTP1 ファイルシステムの内容一覧（TAM サンプル）

ファイル名	使う目的となるファイル	レコード長	レコード数
jnlf01	システムジャーナルファイル	4096 バイト	50 レコード
jnlf02	システムジャーナルファイル	4096 バイト	50 レコード
jnlf03	システムジャーナルファイル	4096 バイト	50 レコード
stsfil01	ステータスファイル	4608 バイト	256 レコード
stsfil02	ステータスファイル	4608 バイト	256 レコード
stsfil03	ステータスファイル	4608 バイト	256 レコード
stsfil04	ステータスファイル	4608 バイト	256 レコード
cpdf01	チェックポイントダンプファイル	4096 バイト	100 レコード
cpdf02	チェックポイントダンプファイル	4096 バイト	100 レコード
cpdf03	チェックポイントダンプファイル	4096 バイト	100 レコード

作成される TAM ファイルの仕様を，次の表に示します。

表 8-9 TAM サンプル用 TAM ファイルの仕様

ファイル名	tamexam1
使う目的となるファイル	TAM ファイル
レコード長	40 バイト（キー長を含む）
キー領域長	20 バイト
キー開始位置	0 バイト目（レコードの先頭）
最大レコード数	10 レコード
テーブル形式	ツリー形式
TAM データファイル名	$DCDIR/examples/tools/tamdata

(4) サンプル UAP の入れ替え

サンプルの UAP は，次に示す手順で入れ替えてください。

1. OpenTP1 システムを停止します。

2. dcsetup コマンドに-d オプションを付けて実行して，いったん OpenTP1 を OS から削除します。
3.「8.4　TAM サンプルの使い方」で示す手順で，使いたいサンプルの UAP を設定し直します。
4. UAP を実行します。

8.5　サンプルプログラムの仕様

ここでは，次に示す 3 種類のサンプルで使う UAP の仕様について説明します。

- Base サンプル
- DAM サンプル
- TAM サンプル

説明するサンプルの仕様は，上記の 3 種類のサンプルで共通です。

8.5.1　サンプルで使うデータベースの内容

サンプルの UAP では，構築した顧客情報のデータベースを，名前をキーにして個人情報を参照したり，販売額を更新したりします。

顧客情報データベースの形式を次の表に示します。

表 8–10　顧客情報データベースの形式

名前	性別	年齢	販売額
Tanaka	Male	25	200000
Saitoh	Female	22	1200000
Nakamura	Male	30	500000
Miyamoto	Male	19	800000
Suzuki	Female	20	950000

8.5.2　サンプルプログラムの処理の概要

サンプルプログラムの処理の概要について説明します。処理については，サンプルプログラムのソースファイルの内容を参照してください。

クライアント UAP は，「参照目的」の要求で，一人の個人情報をサーバのデータベースから取り出します。次に「更新目的」の要求をサーバに送って，販売額を書き替えます。最後に「参照目的」を要求して，更新されたことを確認します。

クライアント UAP からサーバ UAP へサービスを要求するときは，メッセージログを出力して，動作を確認できるようにしています。「参照目的」の要求時には参照後に，「更新目的」の要求時には更新前にメッセージログを出力します。

サーバ UAP 側でも，参照と更新の各処理が完了すると，処理がうまくいったかどうかを知らせるメッセージログが出力されます。

クライアント UAP，サーバ UAP のそれぞれの出力メッセージに"client"，または"server"の文字列が付けられるので，メッセージログがどちらの UAP から出力されたかがわかるようにしてあります。C 言語の場合のクライアント UAP とサーバ UAP の呼び出しの関係を図 8-6 に，COBOL 言語の場合のクライアント UAP とサーバ UAP の呼び出しの関係を図 8-7 に示します。

図 8-6　クライアント UAP とサーバ UAP の呼び出しの関係（C 言語）

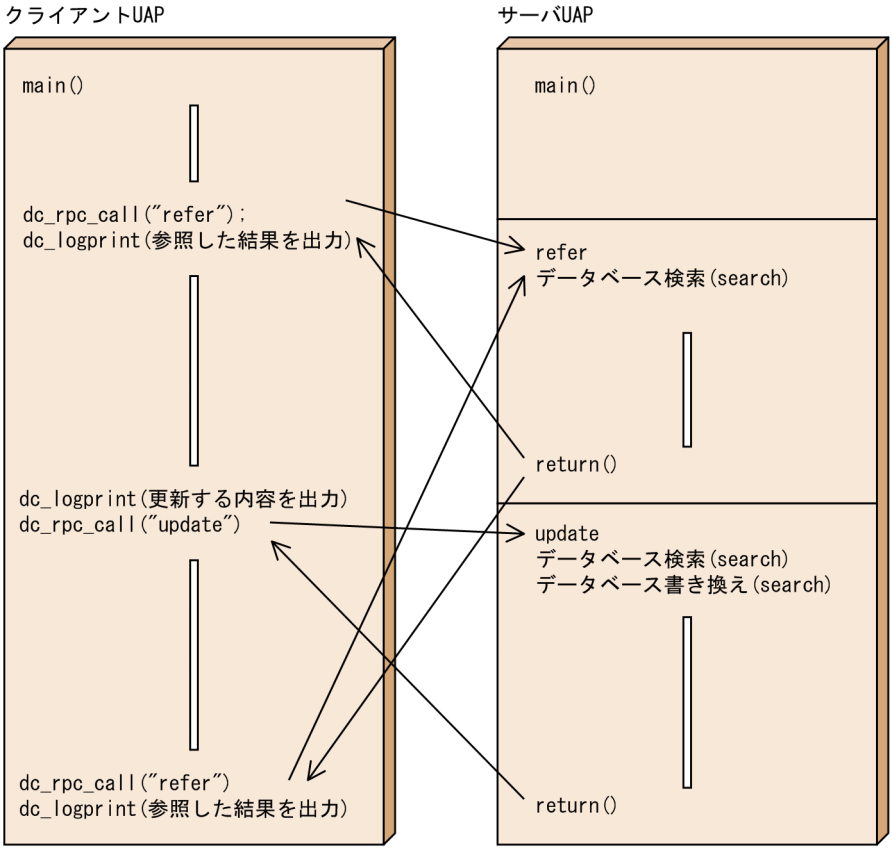
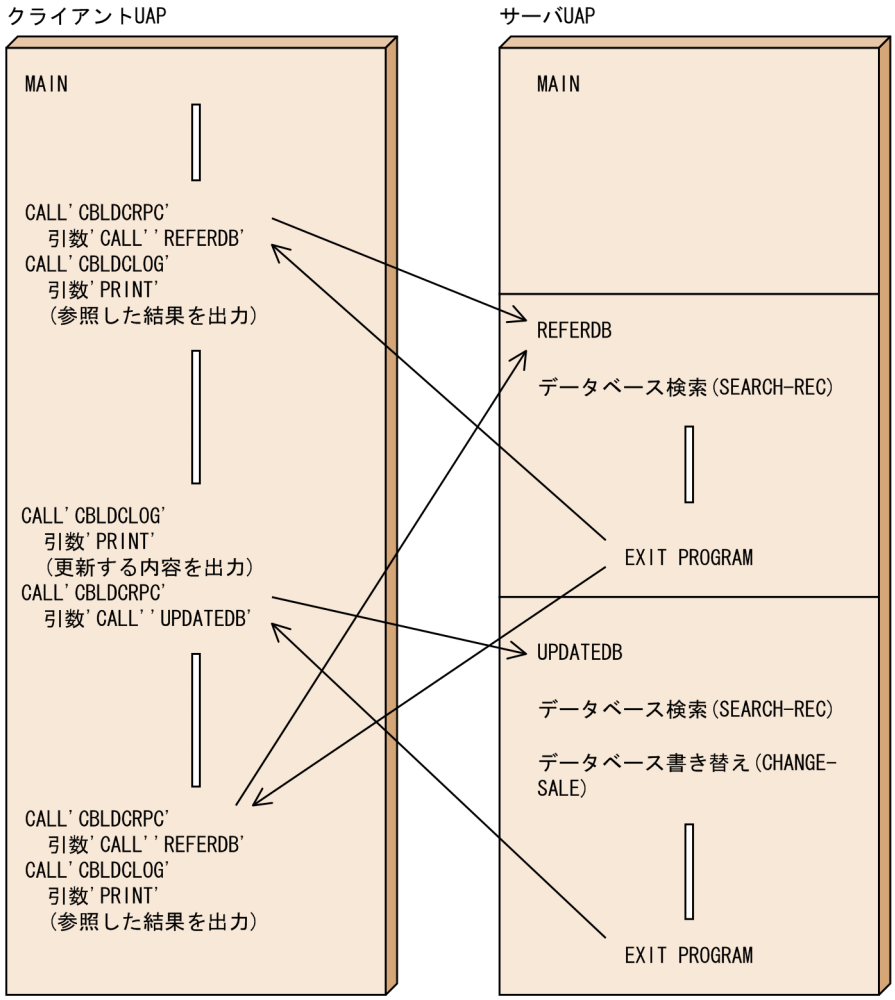


図 8-7　クライアント UAP とサーバ UAP の呼び出しの関係（COBOL 言語）



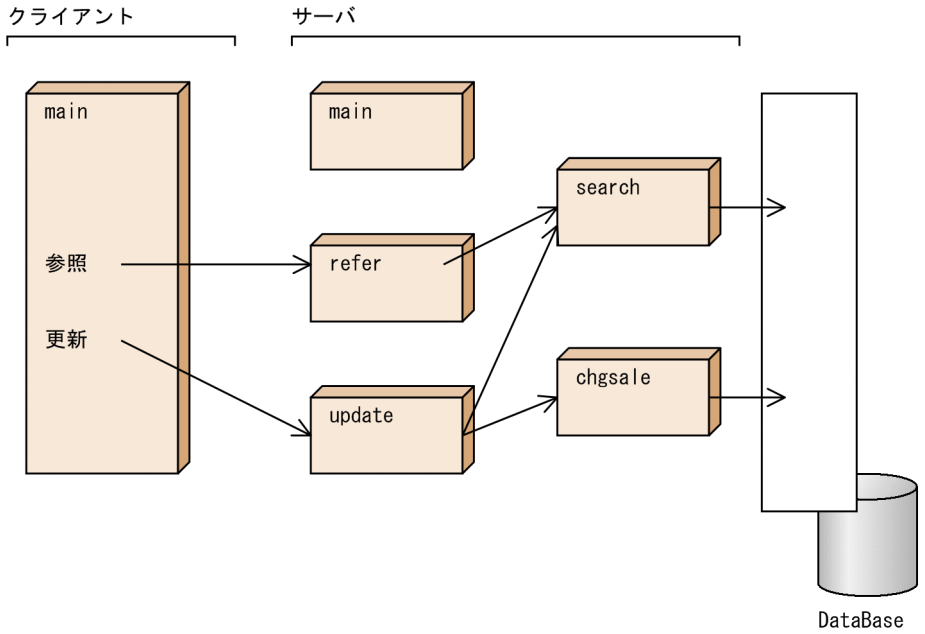
8.5.3　サンプルプログラムのプログラム構造

クライアント UAP は，一つのプログラムから構成されます。サーバ UAP は，複数のプログラムから構成されます。C 言語のサンプル UAP と COBOL 言語のサンプル UAP では，プログラムの名称が一部異なります。

(1)　C 言語のプログラム構造

C 言語の場合の，クライアント UAP とサーバ UAP のプログラム構造を次の図に示します。

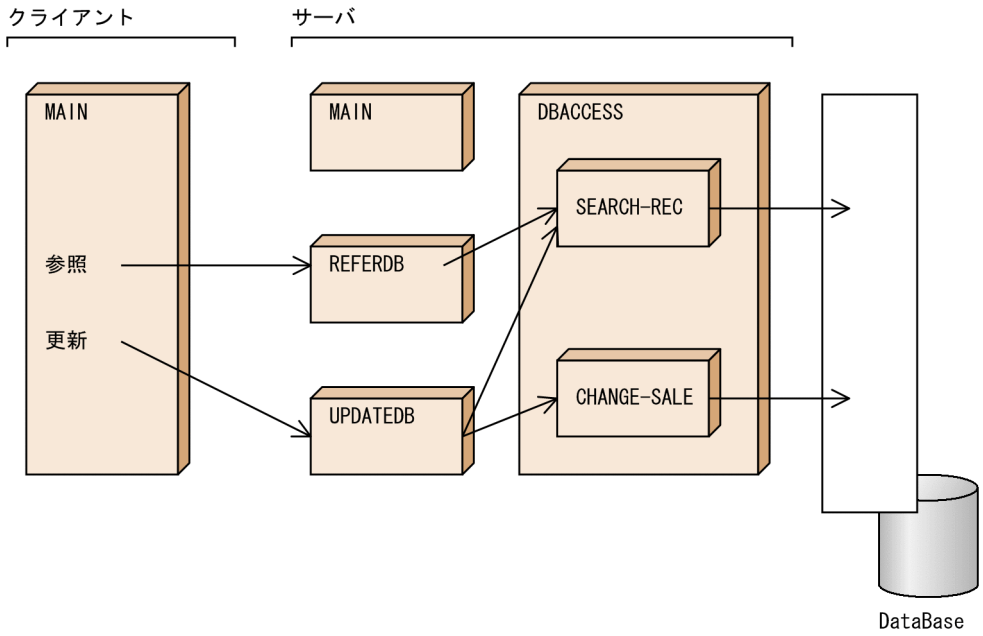
図 8-8　クライアント UAP とサーバ UAP のプログラム構造（C 言語）



(2)　COBOL 言語のプログラム構造

COBOL 言語の場合，サーバ UAP のプログラムの構造で，プログラムを一つ追加しています。Base サンプルの COBOL の UAP のプログラム構造を次の図に示します。

図 8-9　クライアント UAP とサーバ UAP のプログラム構造（COBOL 言語）



8.5.4　サンプル別のプログラムの詳細

サンプル別で異なる仕様について説明します。

(1) Base サンプルのプログラム

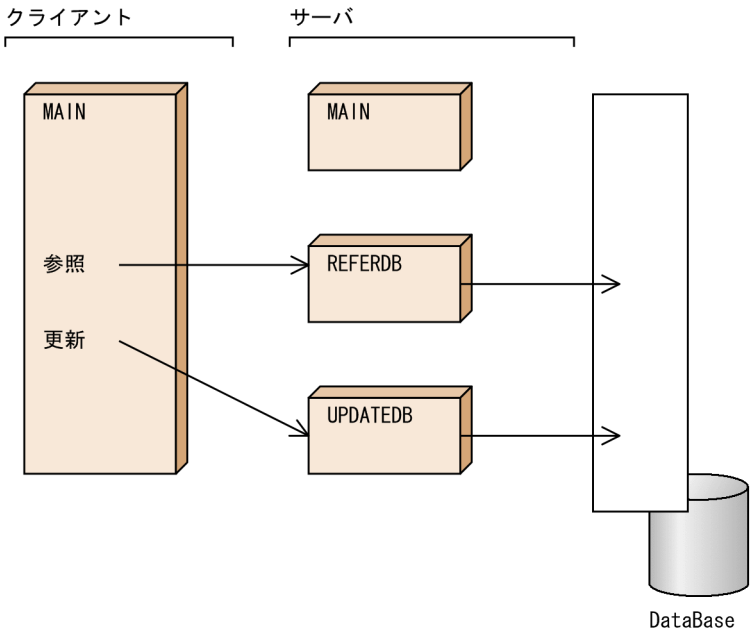
Base サンプルでは，データベース（DataBase）はユーザサーバのプロセスの内部に生成されて，プロセスが常駐している間だけ保持されます。トランザクションの開始と同期点処理は，サーバ UAP 側の更新処理で実行しています。

(2) DAM サンプルのプログラム

次の 3 点を除いて，Base サンプルと同じです。

1. COBOL 言語の場合，サーバ UAP のプログラム構造が，Base サンプルと異なります。DAM サンプルの COBOL の UAP のプログラム構造を次の図に示します。

図 8-10 クライアント UAP とサーバ UAP のプログラム構造（COBOL 言語 DAM サンプル）



2. DAM サンプルは，データベース（DataBase）をオフラインプログラム（dam_mkfs）で作成しているので，ユーザサーバのプロセスが終了してもデータベースは残ります。
3. トランザクションの開始と同期点取得は，サーバ UAP 側の各処理（参照，または更新）で実行しています。

(3) TAM サンプルのプログラム

次の 3 点を除いて，Base サンプルプログラムと同じです。

1. COBOL 言語の場合，サーバ UAP のプログラム構造が，Base サンプルと異なります。TAM サンプルの COBOL の UAP のプログラム構造は，DAM サンプルと同じです。詳細は，図 8-10 を参照してください。
2. TAM サンプルは，データベース（DataBase）をオフラインプログラム（tam_mkfs）で作成しているので，ユーザサーバのプロセスが終了してもデータベースは残ります。

3. トランザクションの開始と同期点取得は，サーバ UAP 側の各処理（参照，または更新）で実行しています。

(4) サンプル使用上の注意

- サンプルを使用して OpenTP1 を起動すると，KFCA00901-W メッセージが出力されることがありますが，使用しないリソースマネジャに関するメッセージの場合は無視してください。
- サンプルプログラムの makefile では C コンパイラとして/bin/cc を明示的に指定しています。/bin/cc がない，または/bin/cc 以外の C コンパイラを使用する場合は，makefile 上で使用する C コンパイラを絶対パスで指定してから使用してください。

8.6 MCF サンプルの使い方

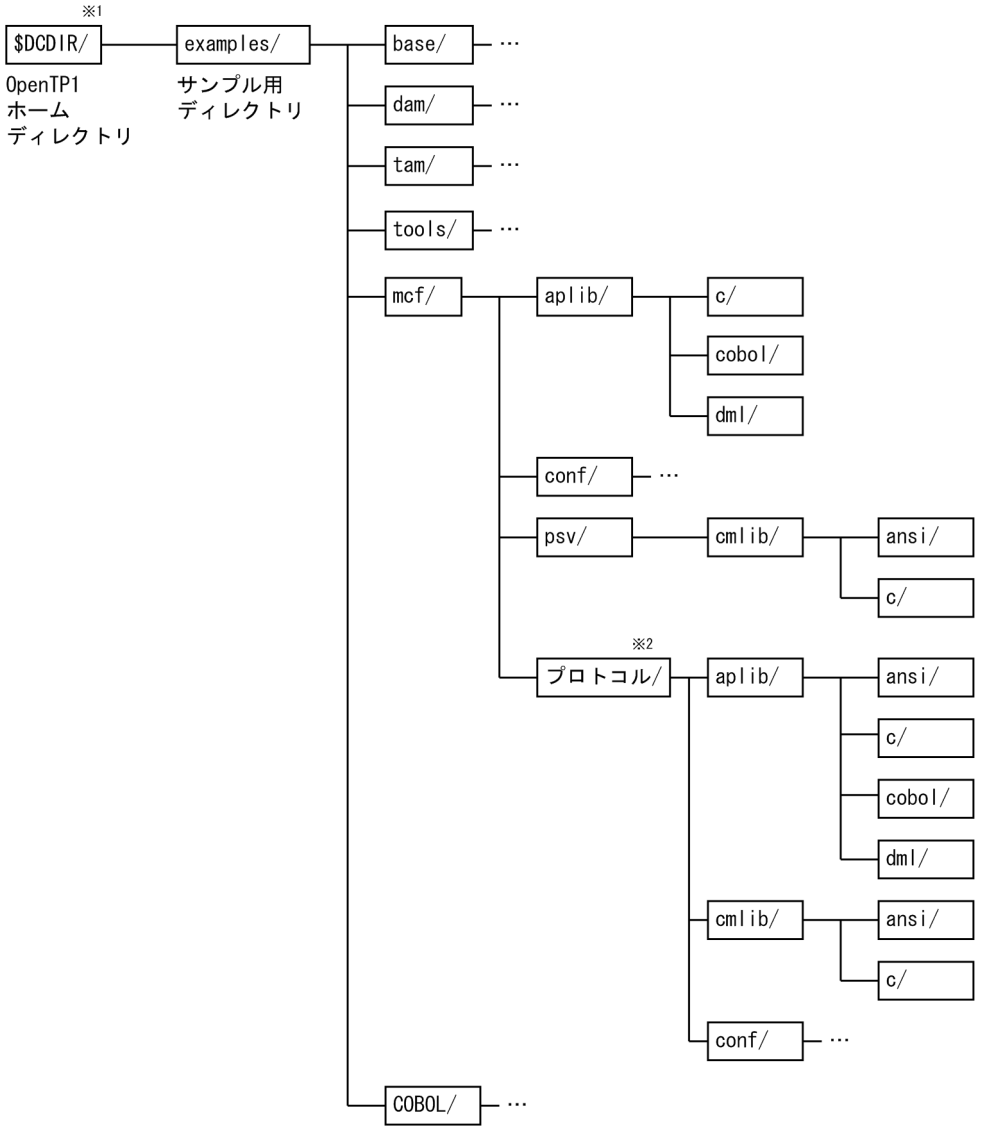
メッセージ制御機能（MCF）のサンプル（**MCF サンプル**）について説明します。MCF サンプルは，マニュアルに掲載している次の記述を，UNIX のテキストファイルとして使えるようにしたものです。

- システム定義例
- MHP のプログラムコーディング例（C 言語，COBOL 言語，DML を使った COBOL 言語）
- MCF メイン関数例（ANSI C，C++の形式と K&R 版 C の形式）

8.6.1 MCF サンプルのディレクトリ構造

MCF サンプルは，MCF の基本部分と通信プロトコル対応部分に分けてあります。MCF サンプルのディレクトリ構造を次の図に示します。

図 8-11　MCF サンプルのディレクトリ構造



注※1 $DCDIRは，OpenTP1ホームディレクトリを示す環境変数です。OpenTP1のサンプル一式をBeTRAN以外のOpenTP1ホームディレクトリにコピーした場合は，そのディレクトリ名を示します。OpenTP1のサンプル一式をコピーしていない場合は，examples/ディレクトリは/BeTRANの下にあります。
注※2 「プロトコル/」ディレクトリには，通信プロトコル製品別で固有なディレクトリ名が付いています。

$DCDIR/examples/mcf/ディレクトリ下の各ディレクトリの内容を次に示します。

aplib/

サンプルの MHP が格納してあるディレクトリ

conf/

サンプルのシステム定義が格納してあるディレクトリ

psv/cmlib/

サンプルの MCF メイン関数が格納してあるディレクトリ

プロトコル/
　通信プロトコル製品別のサンプルが格納してあるディレクトリ

プロトコル／ディレクトリの下にも，次に示すディレクトリが格納してあります。

- サンプルの MHP，SPP が格納してあるディレクトリ
- システム定義が格納されているディレクトリ
- MCF メイン関数が格納されているディレクトリ

(1)　mcf/aplib/ディレクトリの内容

mcf/aplib/ディレクトリの下には，次に示すファイルが格納してあります。

c/
　MHP のソースファイル（C 言語）が格納されているディレクトリです。ここには，MHP のメイン関数（mhp.c）と，MHP のサービス関数（ap1.c）があります。

cobol/
　MHP のソースファイル（COBOL 言語）が格納されているディレクトリです。ここには，MHP のメインプログラム（mhp.cbl）と，MHP のサービスプログラム（ap.cbl）があります。

dml/
　MHP のソースファイル（COBOL 言語 DML）が格納されているディレクトリです。ここには，MHP のサービスプログラム（ap.cbl）があります。

上記のプログラムの内容については，マニュアル「OpenTP1 プログラム作成リファレンス」の該当する言語編のコーディング例を参照してください。

(2)　conf/ディレクトリの内容

mcf/conf/ディレクトリの下には，次に示すファイルが格納してあります。

abc_mngr
　MCF マネジャ定義のサンプルです。

abc_ua_c
　MCF 通信構成定義の共通定義のサンプルです。この定義は，OSAS/UA プロトコル用です。

abc_ua_d
　MCF 通信構成定義のプロトコル固有定義のサンプルです。この定義内容は，OSAS/UA プロトコル用です。

psvr_psvr_cmn
　MCF 通信構成定義の共通定義のサンプルです。この定義は，アプリケーション起動定義用です。

psvr_psvr_dta

MCF 通信構成定義のアプリケーション起動定義のサンプルです。

abc_apli

MCF アプリケーション定義のサンプルです。

mcfu01

MCF システムサービス情報定義のサンプルです。この定義は，MCF 通信プロセス（OSAS/UA 用）の内容です。

mcfu02

MCF システムサービス情報定義のサンプルです。この定義は，アプリケーション通信プロセスの内容です。

上記の定義内容については，マニュアル「OpenTP1 システム定義」の定義例を参照してください。

(3) psv/cmlib/ディレクトリの内容

mcf/psv/cmlib/ディレクトリの下には，次に示すディレクトリが格納してあります。

ansi/

アプリケーション起動プロセス用の MCF メイン関数（ANSI C，C++の形式）のサンプルです。

c/

アプリケーション起動プロセス用の MCF メイン関数（K&R 版 C の形式）のサンプルです。

上記の MCF メイン関数の内容については，マニュアル「OpenTP1 運用と操作」の定義例を参照してください。MCF 通信サービス用の MCF メイン関数については，mcf/プロトコル/cmlib/ディレクトリの下のファイルを参照してください。

(4) プロトコル/ディレクトリについて

プロトコル/ディレクトリの下には，通信プロトコル対応製品別のサンプルが格納してあります。プロトコル/ディレクトリは，通信プロトコル対応製品によって名称が異なります。ディレクトリの名称と製品名の対応を次に示します。

OSASNIF：TP1/NET/OSAS-NIF

OSITP：TP1/NET/OSI-TP

SLUP2：TP1/NET/SLU-TypeP2

TCPIP：TP1/NET/TCP/IP

UDPIP：TP1/NET/UDP

UserAgent：TP1/NET/User Agent

XMAP3：TP1/NET/XMAP3

各ディレクトリの内容については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

8.6.2 MCF サンプルを使うときの注意

MCF サンプルを使うときの注意事項を次に示します。

1. MCF に関連する定義（ネットワークコミュニケーション定義）を使うときは，MCF サンプル内で一貫した定義になるようにしてください。システム定義の内容は，マニュアルに掲載しているものをそのまま使っているため，一部修正が必要な場合があります。また，TP1/Server Base の定義（システムサービス定義）は，MCF サンプルのネットワークコミュニケーション定義に合わせて修正する必要があります。これは，Base サンプルを使う場合も同様です。
2. プロトコル/ディレクトリの下の UAP（MHP，SPP）のサンプルを使う場合，コンパイル／リンケージ時にメイン関数（メインプログラム）が必要です。MHP の場合は，mcf/aplib/ディレクトリにあるメイン関数を修正して使ってください。SPP の場合は，メイン関数を新規で作成してください。

   mcf/conf/ディレクトリの下にある MCF アプリケーション定義（abc_apli）は，作成する MHP に合わせた修正が必要です。さらに，システムサービス定義（ユーザサービス定義など）も，新規で作成する必要があります。
3. MCF メイン関数は，次のサンプルを使ってください。
   - MCF 通信サービスの MCF メイン関数

     /mcf/プロトコル/cmlib/ディレクトリの下のファイル
   - アプリケーション起動サービスの MCF メイン関数

     /mcf/psv/cmlib/ディレクトリの下のファイル

8.7　マルチ OpenTP1 のコマンドを振り分けるサンプル

マルチ OpenTP1 のノードに rsh などを使ってほかのノードからコマンドを実行する場合，どちらの OpenTP1 でコマンドが実行されるかわかりません。そのため，ノード名を指定してコマンドを実行する必要があります。このようなコマンドを実行するために，ノード名を指定してコマンドを実行するコマンド（シェルファイル）がサンプルに格納してあります。このコマンドは，**Base サンプル**の **tools/ディレクトリ**の下に，**delvcmd** という名称で格納してあります。

ただし，Windows 版 TP1/Server Base または TP1/LiNK では，dcmakecon コマンドでコマンドプロンプトを作成し，その中でコマンドを実行してください。dcmakecon コマンドについては，マニュアル「OpenTP1 使用の手引 Windows(R)編」または「TP1/LiNK 使用の手引」を参照してください。

8.7.1　delvcmd コマンドの使い方

delvcmd コマンドは，次の形式で実行します。

```
delvcmd  -w  ノード名〔，ノード名〕… コマンド名
```

ノード名には，同じマシン内のノード識別子を指定します。ノード名は，複数指定できます。複数指定するときは，ノード名とノード名をコンマ（,）で区切ります。

コマンド入力例を次に示します。これは，ノード"nd01"とノード"nd02"に prcls コマンドを実行する例です。入力形式は次に示すどちらでもかまいません。

```
%  delvcmd  ”prcls”  -w  nd01,nd02  <CR>
%  delvcmd  -w  nd01,nd02  ”prcls”  <CR>
```

このコマンドを使う前に，コマンド内にノードごとの$DCDIR，$DCCONFPATH，$SHLIB_PATH，$PATH を，絶対パス名で指定しておいてください。

引数として指定するコマンドは，アポストロフィ（’）と引用符（"）のどちらかで囲んでください。ただし，次に示すコマンドについては制限があります。

- MCF のコマンドの場合はアポストロフィ（’）で囲む。
- TP1/Multi のコマンドの場合は引用符（"）で囲む。

8.7.2　コマンド引数に指定する値の制限

delvcmd コマンドの引数には，**アスタリスク（*）は使えません。**アスタリスクを使ってコマンド引数を一括指定すると，delvcmd コマンドが正常に実行できない場合があります。

delvcmd コマンドに指定するコマンド名には，パイプやリダイレクションは使えません。delvcmd コマンドの実行結果はパイプまたはリダイレクションできます。コマンド入力例を次に示します。

```
% delvcmd "prcls" -w nd01, nd02 > file <CR>
```

8.7.3 delvcmd コマンドで実行できないコマンド

OpenTP1 システムに設定されているアクセス権限によっては，delvcmd コマンドで実行できないコマンドがあります。この場合，delvcmd コマンドを実行する利用者が，対象となるノードの OpenTP1 管理者と同じ利用者名称である必要があります。

8.8 COBOL 言語用テンプレート

UAP を COBOL 言語で作成するときに，**データ部（DATA DIVISION）**のコーディングの負担を軽くするため，**COBOL 言語用テンプレート**を格納してあります。

COBOL 言語用テンプレートは，**/BeTRAN/examples/COBOL/ディレクトリ**の下にあります。

8.8.1 COBOL 言語用テンプレートのファイル

COBOL 言語用テンプレートは，OpenTP1 のシステムサービス別に分けてあります。テンプレートのファイル名は，**DC×××.cbl**（×××は，COBOL-UAP 作成用プログラムの下 3 文字）です。COBOL 言語用テンプレートのファイルを次に示します。

DCADM.cbl：システム運用の管理（CBLDCADM）

DCDAM.cbl：DAM ファイルサービス（CBLDCDAM）

DCDMB.cbl：DAM ファイルサービス（CBLDCDMB）

DCIST.cbl：IST サービス（CBLDCIST）

DCJNL.cbl：ユーザジャーナルの出力（CBLDCJNL）

DCJUP.cbl：ジャーナルデータの編集（CBLDCJUP）

DCLCK.cbl：資源の排他制御（CBLDCLCK）

DCLOG.cbl：メッセージログの出力（CBLDCLOG）

DCMCF.cbl：メッセージ送受信（CBLDCMCF）

DCPRF.cbl：性能検証用トレース（CBLDCPRF）

DCRAP.cbl：リモート API 機能（CBLDCRAP）

DCRPC.cbl：リモートプロシジャコール（CBLDCRPC）

DCRSV.cbl：リモートプロシジャコール（CBLDCRSV）

DCTAM.cbl：TAM ファイルサービス（CBLDCTAM）

DCTRN.cbl：トランザクション制御（CBLDCTRN）

DCUTO.cbl：オンラインテスタの管理（CBLDCUTO）

DCXAT.cbl：アソシエーションの操作（CBLDCXAT）

8.8.2　COBOL 言語用テンプレートの使い方

COBOL 言語用テンプレートを使うときは，次に示す値を**コーディングする UAP の処理に合わせたものに修正する必要**があります。

- データ領域の長さ（ただし，一部のデータだけ）
- 各データ領域へ代入する値

データ領域に設定する値については，マニュアル「OpenTP1 プログラム作成リファレンス COBOL 言語編」の各機能の文法に関する記述を参照してください。

COBOL 言語用テンプレートの使い方には，次に示す 2 とおりがあります。

- **テキストエディタの呼び出し機能を使う方法**
- **COBOL 言語の COPY 文を使う方法**

(1)　テキストエディタの呼び出し機能を使う方法

次の手順でテンプレートを使います。

1. /BeTRAN/examples/COBOL/ディレクトリから，使う機能のテンプレートを選びます。
2. テキストエディタの呼び出し機能を使って，DATA DIVISION 部分を切り取って，UAP のソースプログラムに貼り付けます。
3. 貼り付けた部分を，コーディングの処理に合ったデータ領域に修正します。

(2)　COBOL 言語の COPY 文を使う方法

次の手順でテンプレートを使います。

1. /BeTRAN/examples/COBOL/ディレクトリから，使う機能のテンプレートを選びます。
2. UAP のソースプログラムから，テンプレートのファイル名で COPY 宣言します。このとき，COPY 文に指定するファイル名は，テンプレートのファイル名から「.cbl」を除いた名称を使います。
3. テンプレートのファイルを，COPY 文で参照できるディレクトリに置きます。この方法は，使う COBOL 言語の処理系に従ってください（ファイルのコピー，または環境変数の設定など）。
4. テンプレートのファイルを，コーディングの処理に合ったデータ領域に修正します。

(3)　COBOL 言語用テンプレートを使うときの注意

1. UAP の処理に合わせて変更するデータ領域では，PICTURE 句の長さを（n）と宣言しています。この部分を修正してから使ってください。修正しないままコンパイルすると，エラーになります。
2. 次に示す COBOL 言語用テンプレートは，該当する製品が組み込まれていることが前提になります。
   DCDAM.cbl，DCDMB.cbl：DAM ファイルサービス（CBLDCDAM，CBLDCDMB）

DCTAM.cbl：TAM ファイルサービス（CBLDCTAM）

DCMCF.cbl：メッセージ送受信（CBLDCMCF）

DCUTO.cbl：オンラインテスタの管理（CBLDCUTO）

DCIST.cbl：IST サービス（CBLDCIST）

3. メッセージ送受信のテンプレート（DCMCF.cbl）は，OpenTP1 で使えるすべての MCF 関連の情報が入っています。そのため，通信プロトコル対応製品によっては使えない COBOL-UAP 作成用プログラムのテンプレートがあります。また，データ領域に設定する値も，通信プロトコル対応製品別で異なります。DCMCF.cbl を使う場合は，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編に掲載している COBOL 言語の文法を参照して，適切な形式に変更してから使ってください。
4. UAP の処理に合わせて変更するテンプレートを使う場合は，元のディレクトリからコピーして使うことをお勧めします。

8.9　サンプルシナリオテンプレートの使い方

OpenTP1 ではスケールアウトのシナリオテンプレートを提供しています。このサンプルでは，スケールアウトのシナリオで使用する OpenTP1 設定用スクリプトファイルを利用できます。このサンプルを使用するには，JP1 シナリオ連携の前提となる JP1 製品（JP1/AJS，JP1/AJS2 - Scenario Operation，JP1/Base）が必要です。サンプルシナリオテンプレートの詳細については，マニュアル「OpenTP1 運用と操作」の説明を参照してください。

8.10　リアルタイム取得項目定義テンプレートの使い方

OpenTP1 では，リアルタイム統計情報サービスで使用するテンプレートとして，各種のリアルタイム取得項目定義ファイルを提供しています。これらのファイルはすべて，インストールディレクトリ/rts_template/examples/conf/に格納されます。リアルタイム取得項目定義ファイルは，$DCCONFPATH/直下にコピーして使用します。

リアルタイム取得項目定義ファイルのファイル名と内容を次の表に示します。

表 8-11　リアルタイム取得項目定義ファイルのファイル名と内容

ファイル名	内容
base_itm	BASE 用のリアルタイム取得項目定義ファイル
dam_itm	DAM 用のリアルタイム取得項目定義ファイル
tam_itm	TAM 用のリアルタイム取得項目定義ファイル
all_itm	すべての統計情報を取得するリアルタイム取得項目定義ファイル
none_itm	すべての統計情報を取得しないリアルタイム取得項目定義ファイル
mcfs_itm	MCF 用のリアルタイム取得項目定義ファイル（システム全体，サーバ，サービス単位に取得する項目）
mcfl_itm	MCF 用のリアルタイム取得項目定義ファイル（論理端末単位に取得する項目）
mcfg_itm	MCF 用のリアルタイム取得項目定義ファイル（サービスグループ単位に取得する項目）

リアルタイム取得項目定義の指定方法については，マニュアル「OpenTP1 システム定義」を参照してください。

付録

付録 A　未決着トランザクション情報の出力形式

OpenTP1 の全面回復時に，トランザクションサービス定義の trn_tran_recovery_list オペランドに Y と定義してあれば，**未決着トランザクション情報**をトランザクションサービスのノードにあるディレクトリに出力できます。

付録 A.1　未決着トランザクション情報が出力されるディレクトリとファイル名

未決着トランザクション情報が出力されるディレクトリとファイル名を次に示します。

- 出力先ディレクトリは，トランザクションサービスが存在するノードのディレクトリ「$DCDIR/spool/dctrninf/」に出力されます。
- トランザクションサービスの全面回復が発生するたびに，毎回一つのファイルとして出力されます。ファイル名は「rl ＋トランザクションサービスの開始時間（一意の 8 けたの 16 進数)」となります。

このファイル名は，未決着トランザクション情報を出力したことを知らせるメッセージログに表示されます。

不要になった未決着トランザクション情報ファイルは，削除してください。削除する方法を次に示します。

- **コマンドで削除する方法**
  trndlinf コマンドを実行します。
- **OpenTP1 の開始時に，前回までのオンラインで作成した情報を削除する方法**
  トランザクションサービス定義の trn_recovery_list_remove オペランドと trn_recovery_list_remove_level オペランドに，削除する条件を指定しておきます。

付録 A.2　未決着トランザクション情報の出力内容

未決着トランザクション情報として，次の項目が出力されます。

1. OpenTP1 システムノード ID
   OpenTP1 のシステムノード ID。
2. グローバルトランザクション番号
   グローバルトランザクションを管理するためにシステムで一意に付けた番号。
3. トランザクションブランチ番号
   トランザクションブランチを管理するためにシステムで一意に付けた番号。
4. トランザクション第 1 状態
   トランザクションブランチの処理状態。

5. トランザクション第 2 状態
   トランザクションブランチのプロセスに関する状態。
6. トランザクション第 3 状態
   トランザクションブランチの通信状態。
7. プロセス ID
   トランザクションブランチが動作しているプロセスのプロセス ID。
8. サーバ名
   トランザクションブランチを起動しているサーバの名称。
9. サービス名
   トランザクションブランチを起動しているサービスの名称。
10. トランザクション記述子
    同一トランザクショングローバル識別子を持つトランザクションブランチを区別するためのインデクス番号。
11. ブランチ記述子
    一つのトランザクションブランチから分岐したトランザクションブランチを区別するためのインデクス番号。ルートトランザクションブランチの場合は「**********」が表示されます。
12. 親トランザクション記述子
    該当するトランザクションブランチを生成したトランザクションのトランザクション記述子。ルートトランザクションブランチの場合は「**********」が表示されます。

付録 A.3　未決着トランザクション情報の出力形式

未決着トランザクション情報の出力形式と出力例を図 A-1 と図 A-2 に示します。

図 A-1　未決着トランザクション情報の出力形式

```
                                                                                                     ①
Undecided transaction information                                         mmm  dd  HH:MM:SS yyyy

TRNGID          TRNBID          状態              PID サーバ サービス  ENTRYID   BRANCHID   PENTRYID
aaaaaaaabbbbbbbb aaaaaaaacccccccc dd··dd(e,f) gggggggggg hh··hh ii··ii jjjjjjjjjj kkkkkkkkkk llllllllll    ②
aaaaaaaabbbbbbbb aaaaaaaacccccccc dd··dd(e,f) gggggggggg hh··hh ii··ii jjjjjjjjjj kkkkkkkkkk llllllllll
                                                  :
                                                  :
```

（説明）
①全面回復を開始した時刻
　mmm：月（英小文字）　dd：日　HH：時　MM：分　SS：秒　yyyy：西暦（d，H，M，S，yは数字）
②トランザクション情報
　aaaaaaaa　：OpenTP1のシステムノードID（8文字）
　bbbbbbbb　：グローバルトランザクション番号（16進文字列）
　cccccccc　：トランザクションブランチ番号（16進文字列）
　dd…dd　：トランザクション第1状態（20文字以内）
　　BEGINNING　… トランザクションブランチ開始処理中状態
　　ACTIVE　… 実行中状態
　　SUSPENDED　… 中断中状態
　　IDLE　… 同期点処理へ移行状態
　　PREPARE　… コミット（1相目）処理中状態
　　READY　… コミット（2相目）処理待ち状態
　　HEURISTIC_COMMIT　… ヒューリスティック決定コミット処理中状態
　　HEURISTIC_ROLLBACK　… ヒューリスティック決定ロールバック処理中状態
　　COMMIT　… コミット処理中状態
　　ROLLBACK_ACTIVE　… ロールバック処理待ち状態
　　ROLLBACK　… ロールバック処理中状態
　　HEURISTIC_FORGETTING … ヒューリスティック決定後のトランザクションブランチ終了処理中状態
　　FORGETTING　… トランザクションブランチ終了処理中状態
　e　：トランザクション第2状態（1文字）
　　u … ユーザサーバプロセスでのユーザサーバ実行中状態
　　r … トランザクション回復プロセスでのトランザクションブランチ回復処理実行中状態
　　p … トランザクション回復プロセスでの他トランザクションブランチの回復処理完了待ち状態
　　なお，第1状態がREADYでルートトランザクションブランチが同一ノード内にない場合は，ユーザの指示待ち状態
　f　：トランザクション第3状態（1文字）
　　s … 送信中です
　　r … 受信中です
　　n … 送受信中ではありません
　　なお，送受信中とは，トランザクションマネジャによるトランザクションブランチ間の同期合わせのことです。
　gggggggggg：プロセスID（10進数）
　hh … hh　：サーバ名（8文字以内）
　ii … ii　：サービス名（32文字以内）　ただし，SUPの場合は空白
　jjjjjjjjjj：トランザクション記述子（10進数）
　kkkkkkkkkk：ブランチ記述子（10進数）※
　llllllllll：親トランザクション記述子（10進数）※

注※　ルートトランザクションブランチの場合は「**********」を表示します。

図 A-2　未決着トランザクション情報の出力例

```
Undecided transaction information                  Jun 10 12:43:55  1995

TRNGID          TRNBID          状態              PID サーバ サービス    ENTRYID   BRANCHID   PENTRYID
@@@@hst100000001 @@@@hst100000001 COMMIT(p,n)        0 sup1                  0 ********** **********
@@@@hst100000001 @@@@hst100000006 COMMIT(p,n)        0 spp1   sv1            1          0          0
@@@@hst100000002 @@@@hst10000000a ROLLBACK(p,n)      0 sup2                  2 ********** **********
@@@@hst100000002 @@@@hst10000000e ROLLBACK(p,n)      0 spp2   sv1            3         32          2
                              :
                              :
```

付録 B　デッドロック情報の出力形式

複数の UAP 間でデッドロックが起こった場合，ロックサービス定義の lck_deadlock_info オペランドに Y と指定してあれば，デッドロック情報をロックサービスのノードにあるディレクトリに出力します。デッドロック情報を出力するのは次の場合です。

- ロックサービスがデッドロックを検知した場合（**デッドロック情報**）
- 排他解除待ちで，タイムアウトになった場合（**タイムアウト情報**）

付録 B.1　デッドロック情報が出力されるディレクトリとファイル名

デッドロック情報は，次に示す形式で出力されます。

- デッドロック情報の出力先ディレクトリは，デッドロックまたはタイムアウトを検知したロックサービスが存在するノードのディレクトリ「$DCDIR/spool/dclckinf/」に出力されます。
- デッドロック情報が発生するたびに，毎回一つのファイルとして出力されます。

  ファイル名は，デッドロックまたはタイムアウトが起こった日付と時刻がファイル名になります。ファイル名の長さは日付が 1 けたか 2 けたかによって異なります。

  （例）

  10 月 3 日 7 時 41 分 00 秒のとき…Oct3074100

  10 月 10 日 15 時 5 分 27 秒のとき…Oct10150527

  このファイル名は，デッドロックが起こったことを知らせるメッセージログに表示されます。不要となったファイルは削除してください。

付録 B.2　デッドロック情報の出力形式

デッドロックを検知したときのデッドロック情報の出力形式と出力例を図 B-1 と図 B-2 に示します。

図 B-1　デッドロック情報の出力形式

```
                                                                        ①
    Deadlock  information                                        Jul DD HH:MM:SS YYYY

 ②  server:CCCCCCCC    pid:CCCCC
 ③  GID:CCCCCCCCCCCCCCCC
 ④  BID:CCCCCCCCCCCCCCCC
     occupy
 ⑤    server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC
     wait
 ⑥    server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC
 ⑦   wait start time HH:MM:SS

    server:CCCCCCCC    pid:CCCCC
    GID:CCCCCCCCCCCCCCCC
    BID:CCCCCCCCCCCCCCCC
     occupy
      server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC
     wait
      server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC
     wait start time HH:MM:SS
                                      :
                                      :
                                      :
```

（説明）

① デッドロックを検出した時刻
② アクセス要求元のサーバ名とプロセスID
③ アクセス要求元のトランザクショングローバル識別子
④ アクセス要求元のトランザクションブランチ識別子
⑤ 資源を確保しているサーバの情報
- 排他要求サーバ名
- 排他制御モード（ PR または EX ）
- 占有している資源名称
- MIGRATE/BRANCHの要求種別

⑥ 資源を解放待ちしているサーバの情報
- 排他要求サーバ名
- 排他制御モード（ PR または EX ）
- 解放待ちの資源名称
- MIGRATE/BRANCHの要求種別

⑦ 資源の解放待ちが起こった時刻

注　②～⑦の情報が，デッドロックを構成していたすべてのUAPプロセスについて出力されます。

図 B-2　デッドロック情報の出力例

```
Deadlock  information                                        Jan 27 14:41:43 1995

server:spp2        pid:2076
GID:@@@@nd0100000001
BID:@@@@nd0100000002
 occupy
  server name:usr       mode:EX  resource name:file2            owner:BRANCH
 wait
  server name:          mode:    resource name:                 owner:
 wait start time  :  :

server:sup2        pid:2078
GID:@@@@nd0100000001
BID:@@@@nd0100000001
 occupy
  server name:          mode:    resource name:                 owner:        ]※
 wait
  server name:usr       mode:EX  resource name:file1            owner:BRANCH
 wait start time 14:41:40

server:sup3        pid:2079
GID:@@@@nd0100000012
BID:@@@@nd0100000020
 occupy
  server name:usr       mode:EX  resource name:file1            owner:BRANCH
 wait
  server name:usr       mode:EX  resource name:file2            owner:BRANCH
 wait start time 14:41:27
```

注※　占有している資源がない場合は，空欄になります。

付録 B.3　タイムアウト情報の出力形式

タイムアウトを検知したときのタイムアウト情報の出力形式と出力例を図 B-3 と図 B-4 に示します。

図 B-3　タイムアウト情報の出力形式

```
                                                                      ①
    Timeout  information                                          Jul DD HH:MM:SS YYYY

 ②  server:CCCCCCCC  pid:CCCCC
 ③  GID:CCCCCCCCCCCCCCCC
 ④  BID:CCCCCCCCCCCCCCCC
     occupy
 ⑤    server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC
      server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC
                                  :
                                  :
     wait
 ⑥    server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC

 ⑦  server:CCCCCCCC    pid:CCCCC
 ⑧  GID:CCCCCCCCCCCCCCCC
 ⑨  BID:CCCCCCCCCCCCCCCC
     occupy
 ⑩    server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC

    server:CCCCCCCC  pid:CCCCC
    GID:CCCCCCCCCCCCCCCC
    BID:CCCCCCCCCCCCCCCC
     occupy
      server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC

    server:CCCCCCCC  pid:CCCCC
    GID:CCCCCCCCCCCCCCCC
    BID:CCCCCCCCCCCCCCCC
     occupy
      server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC

    server:CCCCCCCC  pid:CCCCC
    GID:CCCCCCCCCCCCCCCC
    BID:CCCCCCCCCCCCCCCC
     occupy
      server name:CCCCCCCC  mode:CC  resource name:CCCCCCCCCCCCCCCC  owner:CCCCCCC
                                 :
                                 :
                                 :
```

（説明）
① タイムアウトを検出した時刻
② タイムアウトになったサーバ名とプロセスID
③ タイムアウトになったトランザクショングローバル識別子
④ タイムアウトになったトランザクションブランチ識別子
⑤ タイムアウトになったサーバが占有していた情報。タイムアウトになったサーバがその時点で占有していたすべての資源について出力されます。
・排他要求サーバ名
・排他制御モード（ PR または EX ）
・占有している資源名称
・MIGRATE/BRANCHの要求種別
⑥ タイムアウトになったサーバの待ち情報
・排他要求サーバ名
・排他制御モード（ PR または EX ）
・解放待ちの資源名称
・MIGRATE/BRANCHの要求種別
⑦ タイムアウトの要因を構成しているサーバ名とプロセスID
⑧ タイムアウトの要因を構成しているトランザクショングローバル識別子
⑨ タイムアウトの要因を構成しているトランザクションブランチ識別子
⑩ タイムアウトの要因を構成しているサーバが占有していた情報。タイムアウトの要因を構成していたすべてのサーバについて出力されます。
・排他要求サーバ名
・排他制御モード（ PR または EX ）
・占有している資源名称
・MIGRATE/BRANCHの要求種別

注　タイムアウトが起こった時点で，タイムアウトとなったサーバが占有しようとした資源に対して確保／解放待ちをしていたサーバすべての情報（⑦～⑩）が出力されます。

図 B-4　タイムアウト情報の出力例

```
Timeout  information                                        Jun 30 16:25:45 1995


server:sup1        pid:2940
GID:@@@@nd0100000008
BID:@@@@nd0100000008
 occupy
  server name:usr       mode:PR  resource name:fileC            owner:BRANCH     ┐
  server name:usr       mode:EX  resource name:fileB            owner:BRANCH     ┘※
  server name:usr       mode:PR  resource name:fileE            owner:BRANCH
 wait
  server name:usr       mode:EX  resource name:fileA            owner:BRANCH

server:sup3        pid:2944
GID:@@@@nd0100000003
BID:@@@@nd0100000004
 occupy
  server name:usr       mode:PR  resource name:fileA            owner:BRANCH

server:sup2        pid:2941
GID:@@@@nd0100000004
BID:@@@@nd0100000005
 occupy
  server name:usr       mode:PR  resource name:fileA            owner:BRANCH
```

注※　占有している資源がない場合は，空欄になります。

付録 B.4　TP1/FS/Table Access を使用した場合の出力形式

TP1/FS/Table Access を使用していて，その資源でデッドロック・タイムアウトが発生した場合は，テーブル名やキー値などの情報も出力されます。

デッドロックを検知したときのデッドロック情報の出力例を次の図に示します。

図 B-5　TAM 資源のデッドロック情報の出力例

```
      Deadlock information                                        Dec 15 17:26:30 1995
       server:sups01x     pid:7786
       GID:b81eBR0S00000004
       BID:BR0SBR0S00000004
        occupy
         server name:_tam      mode:EX  resource name:R000010000000523  owner:MIGRATE
  ①                  --> TAM Table name = [tam_primary_table]
  ②                  --> TAM Record Key (Length=[10])
  ③              [ 30 30 30 30 30 30 30 35 32 32 00 00 00 00 00 00 ]:0000000522......
        wait
         server name:_tam      mode:EX  resource name:R000010000000522  owner:MIGRATE
                    --> TAM Table name = [tam_primary_table]
                    --> TAM Record Key (Length=[10])
                [ 30 30 30 30 30 30 30 35 32 31 00 00 00 00 00 00 ]:0000000521......
        wait start time 17:26:28
       server:sups01t     pid:7781
       GID:b81dBR0S00000003
       BID:BR0SBR0S00000003
        occupy
         server name:_tam      mode:EX  resource name:T00003            owner:MIGRATE
                    --> TAM Table name = [tam_primary_table]
                    --> TAM Record Key (Length=[10])
                [ 30 30 30 30 30 30 30 35 32 31 00 00 00 00 00 00 ]:0000000521......
        wait
         server name:_tam      mode:EX  resource name:T00001            owner:MIGRATE
                    --> TAM Table name = [tam_primary_table]
                    --> TAM Record Key (Length=[10])
                [ 30 30 30 30 30 30 30 35 32 32 00 00 00 00 00 00 ]:0000000522......
        wait start time 17:26:28
```

(説明)

① 'resource name'の資源名称が示しているTAMテーブル名称
　資源名称が'T'で始まる場合は①で示すTAMテーブルが資源です。②，③は出力されません。
　資源名称が'R'で始まる場合はTAMレコードが資源で，①で示すTAMテーブル上にあります。②，③が出力されます。

② 'resource name'の資源名称が示しているTAMレコードのキー長

③ 'resource name'の資源名称が示しているTAMレコードのキー値の内容
　'[]'内には16進数表示でキー値を出力しています。
　その右側には，印字できる文字であればそのキャラクタを，印字できない文字であれば'.'を出力しています。キー値が16の整数倍に満たない場合は，'[]'内の余りには'00'が出力されます。

付録 C　マルチスケジューラ機能の検討が必要なシステム構成例

システムの大規模化，マシンやネットワークの高性能化などに伴って，従来のスケジューラだけでは効率良くスケジューリングできないことがあります。ここでは，マルチスケジューラ機能の検討が必要なシステム構成例とその解決例について説明します。

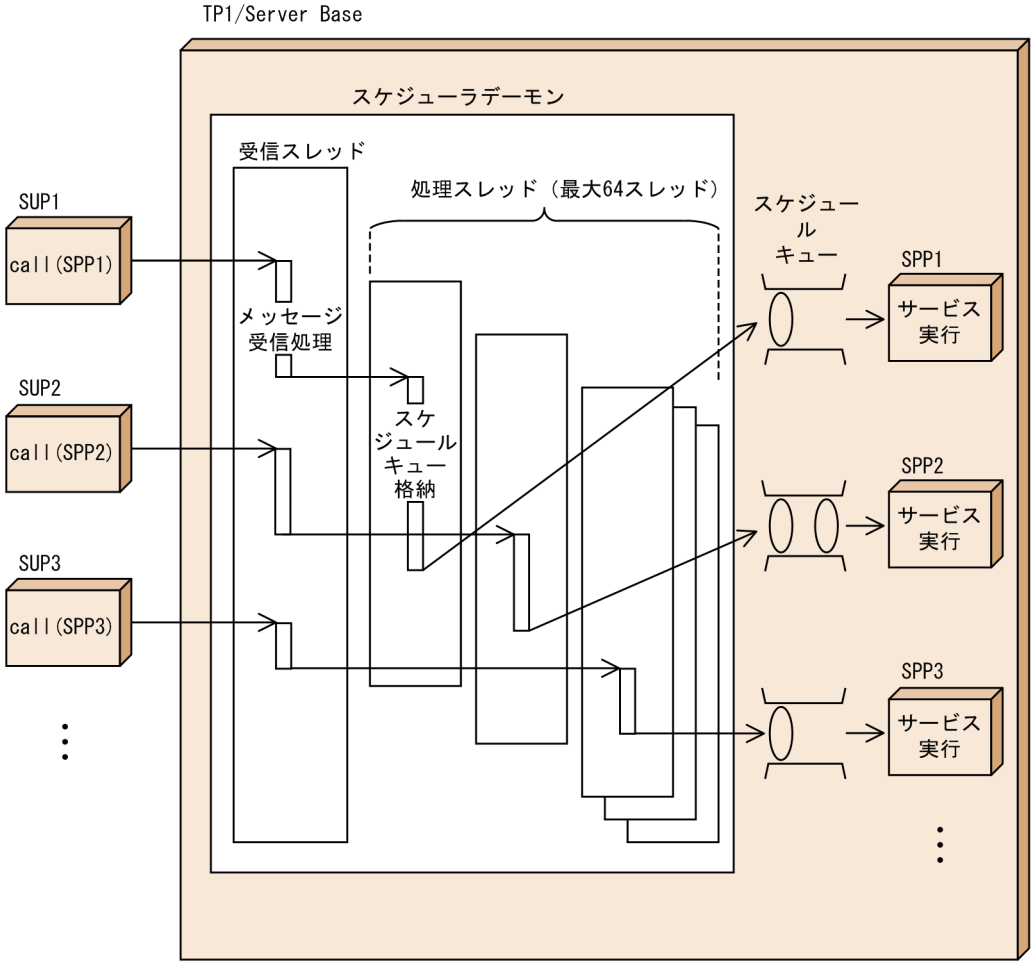
付録 C.1　スケジューラ機能の処理概要

スケジューラでは，クライアント UAP から他ノードのキュー受信型サーバ（スケジュールを使う SPP）にサービスを要求した場合，要求先サーバが存在するノードのスケジューラデーモンが，いったんサービス要求メッセージを受信して，該当するキュー受信サーバのスケジュールキューに格納します。

スケジューラデーモンは，クライアント UAP からのサービス要求メッセージを受信する受信スレッド（1 スレッド）と，サービス要求をスケジュールへ格納する処理スレッド（最大 64 スレッド）で構成されています。

スケジューラ機能の処理概要を次の図に示します。

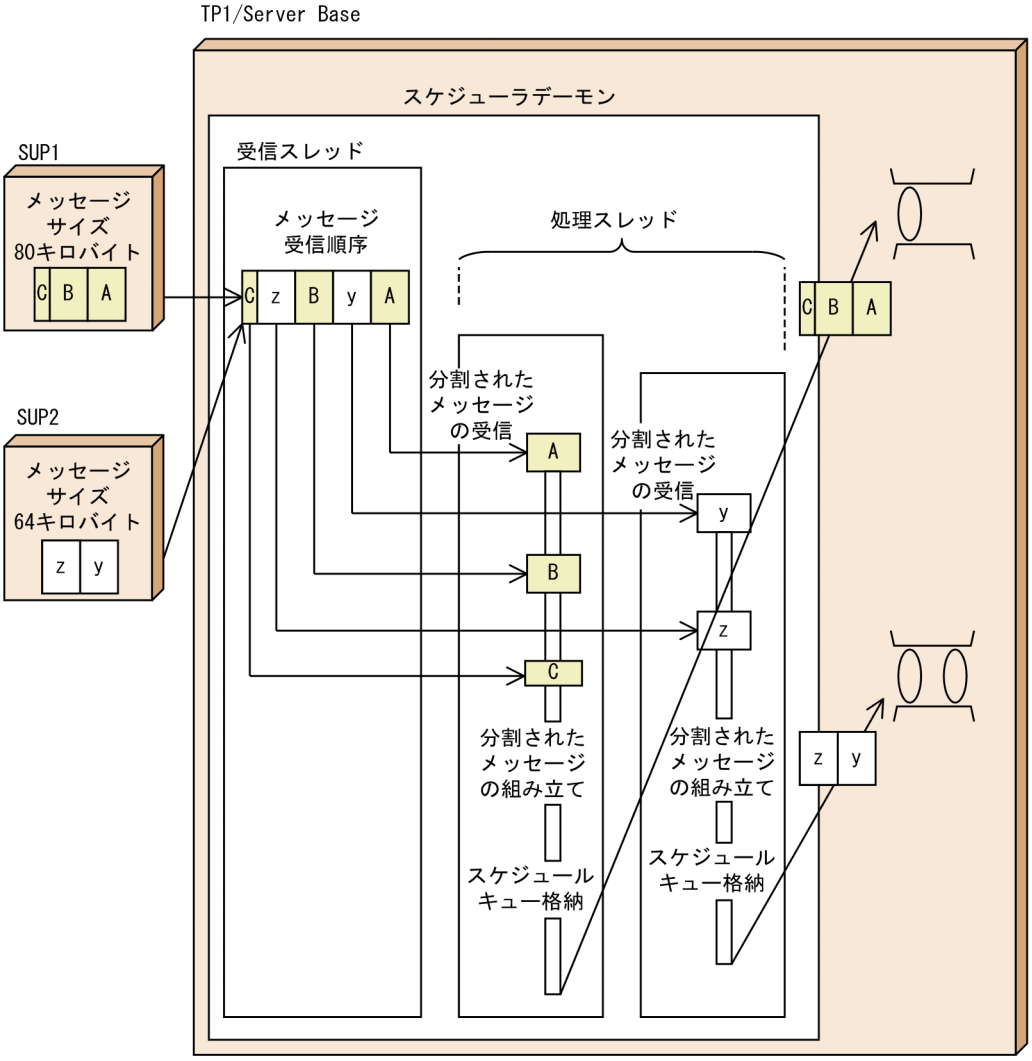
図 C-1　スケジューラ機能の処理概要



スケジューラデーモンの受信スレッドは，クライアント UAP からのサービス要求メッセージのうち，一度に受信できるメッセージ長を 32 キロバイトまでとしています。32 キロバイトを超えたサービス要求メッセージの場合は，メッセージを分割して送受信することによって，スケジューラデーモンが 1 クライアント UAP のメッセージ受信処理に占有されないようにしています。

サービス要求メッセージの処理概要を次の図に示します。

図 C-2　サービス要求メッセージの処理概要



この図に示すように，スケジューラは，クライアント UAP からのサービス要求メッセージをスケジューリングします。

付録 C.2　スケジューラが原因となるおそれのあるシステム構成例

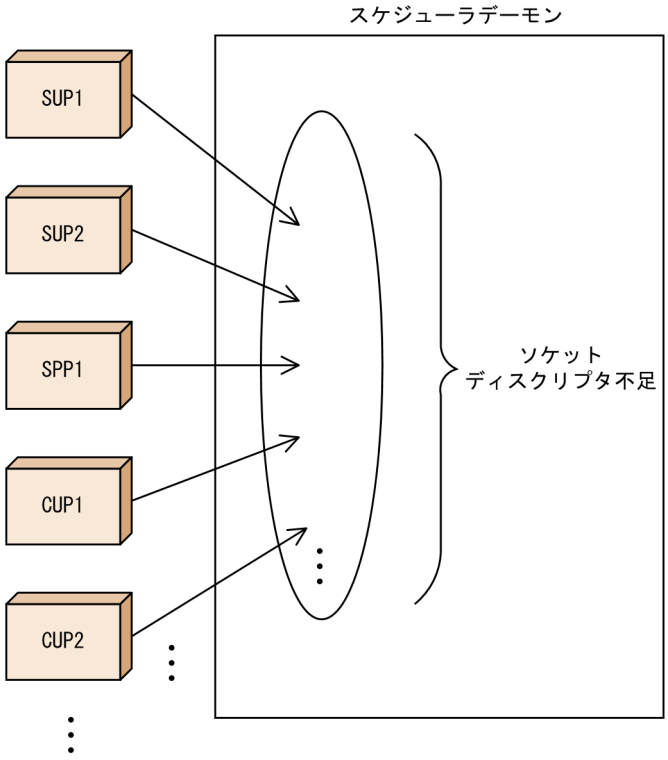
システムの大規模化，マシンやネットワークの高性能化などに伴って，従来のスケジューラデーモンだけでは効率良くスケジューリングできない場合があります。スケジューラが原因となるおそれのあるシステム構成例について次に示します。

(1) ソケットディスクリプタが不足するシステム

1 スケジューラデーモンに接続するクライアント UAP 数が増加すると，スケジューラデーモンが利用するソケットディスクリプタ数に，余裕を持った値を指定できないことがあります。スケジューラデーモンでソケットディスクリプタが不足すると，新たなソケットディスクリプタを確保するために接続済みのコネクションに対して，切断要求および切断処理が発生します。このコネクション切断処理を行うためにシステムにかかる負荷によって，スケジューラデーモンのスケジューリング性能が低下する場合があります。

ソケットディスクリプタが不足するシステム例を次の図に示します。

図 C-3　ソケットディスクリプタが不足するシステム例



(2) connect システムコールがエラーになるシステム

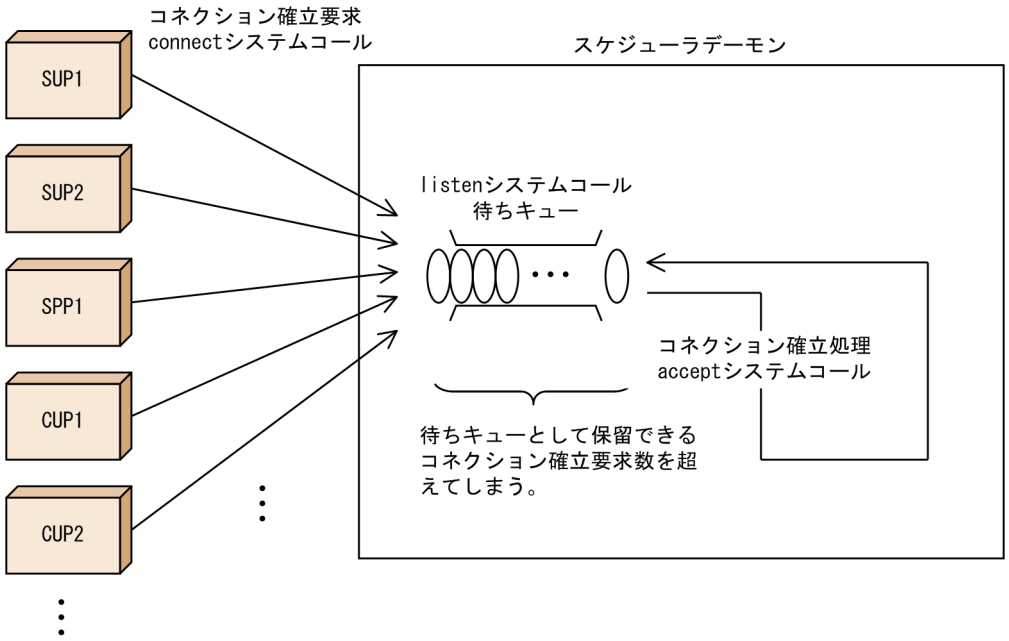
OpenTP1 では通信プロトコルに TCP/IP を利用しているため，クライアント UAP からのコネクション確立要求（connect システムコール）は，accept システムコールによって取り出されるまで listen システムコールの待ちキューとして保留されます。

コネクション確立要求を保留する待ちキューの数は，OS によって異なります。しかし，クライアント UAP からの要求が集中した場合，キューとして保留できる数を超えてコネクション確立要求が発生することがあります。

待ちキューとして保留できないコネクション確立要求が発生すると，CUP（TP1/Client）の場合はメッセージ KFCA02449-E を，SUP および SPP（TP1/Server Base）の場合はメッセージ KFCA00327-W を出力します。そしてこのサービス要求は通信障害または OpenTP1 未起動として扱われる場合があります。

connect システムコールがエラーになるシステム例を次の図に示します。

図 C-4　connect システムコールがエラーになるシステム例



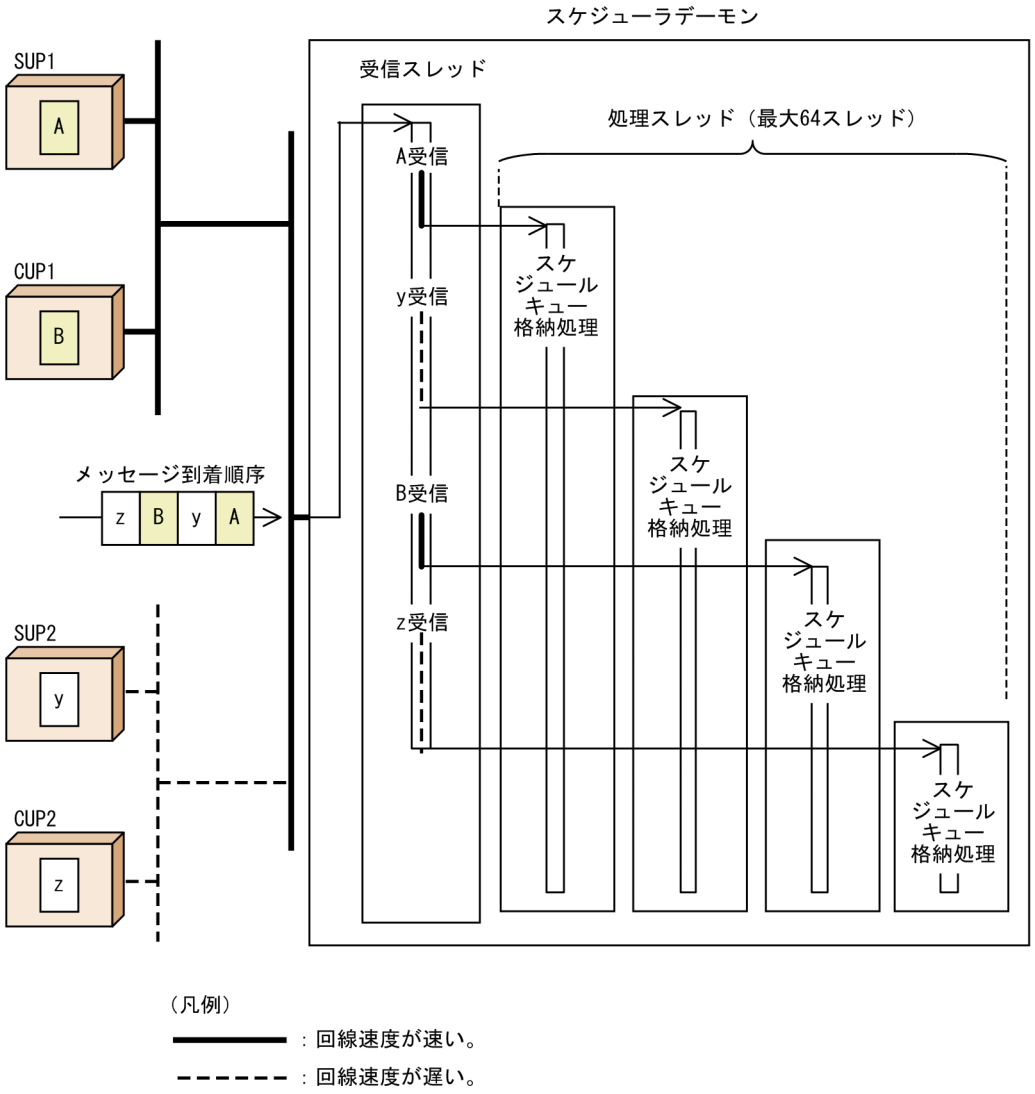
(3)　回線速度が異なるネットワークを混在して利用しているシステム

スケジューラデーモンの受信スレッドは，特定のクライアント UAP のサービス要求メッセージを受信している間は，ほかのクライアント UAP のサービス要求メッセージを受信できません（ただし，サービス要求メッセージが 32 キロバイトを超える場合は分割して送受信されます）。

このため，回線速度が遅いネットワークに接続しているクライアント UAP のメッセージ受信処理によって，回線速度が速いネットワークに接続しているクライアント UAP からのメッセージ受信処理が遅延して，回線速度が速いネットワークの性能を有効に利用できない場合があります。

回線速度が異なるネットワークを混在して利用しているシステム例を次の図に示します。ここでは，回線速度が異なる二つの回線を比較した場合の例を挙げています。2 倍の回線速度差がある回線を比較すると，受信処理にも回線速度と同じ 2 倍の時間差が出ることを示しています。

図 C-5　回線速度が異なるネットワークを混在して利用しているシステム例

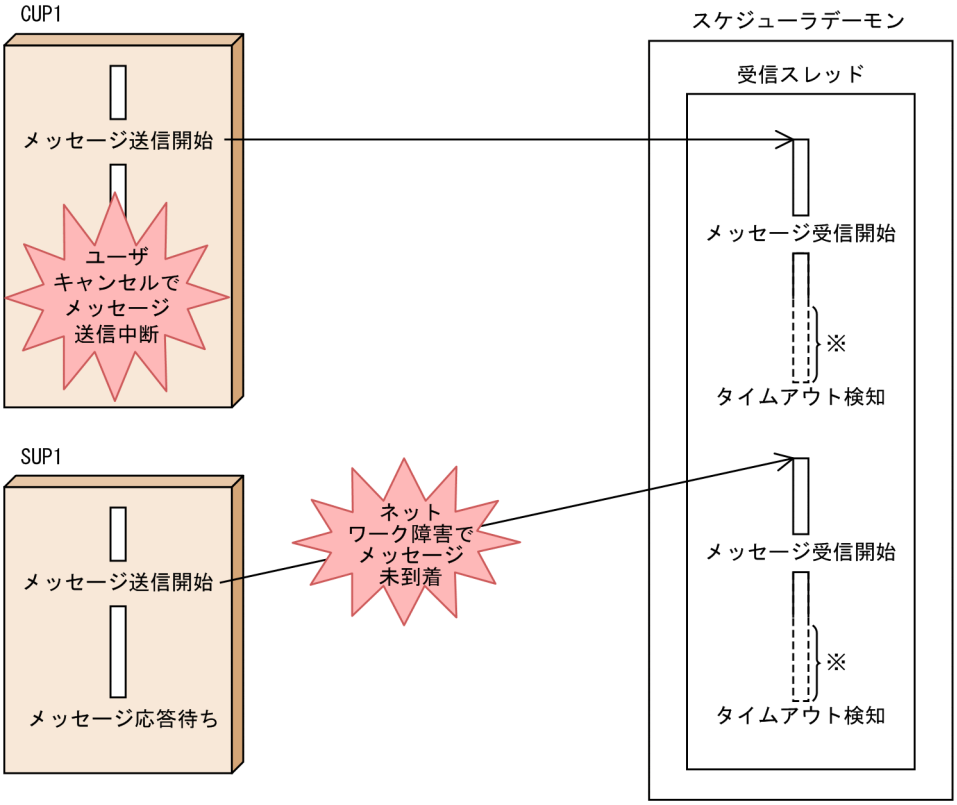


(4)　サービス要求メッセージが途切れるシステム

クライアント UAP が強制終了するなどの理由によって，スケジューラデーモンへのサービス要求メッセージの送信処理が途切れると，受信スレッドのメッセージ受信処理がタイムアウトするまでスケジューリングが滞る場合があります。

サービス要求メッセージが途切れるシステム例を次の図に示します。

図 C-6　サービス要求メッセージが途切れるシステム例



注※　タイムアウトを検知するまでは，受信処理は実行されません。

(5)　処理スレッドが一時的に不足するシステム

クライアント UAP からのサービス要求メッセージが極端に集中して，スケジューラデーモンの処理能力を超えてしまうと，一時的に処理スレッドが不足することがあります。

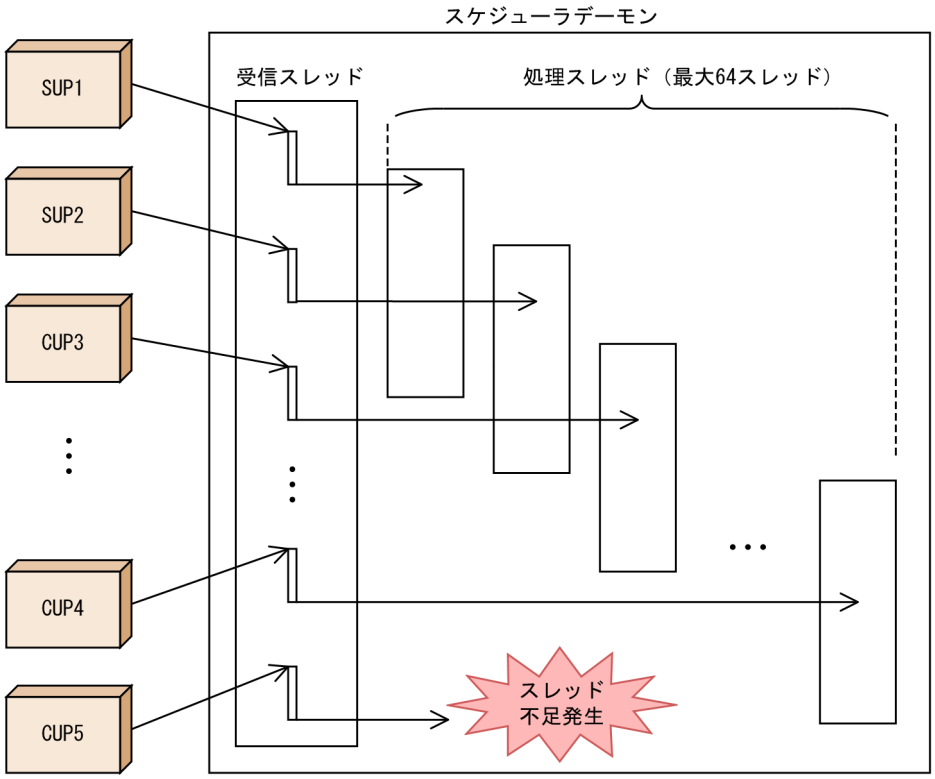
処理スレッド不足が発生すると，メッセージ KFCA00356-W を出力して，一時的にサービス要求が通信障害やタイムアウトとして扱われる場合があります。

メッセージ KFCA00356-W は，システム共通定義の rpc_server_busy_count によって出力するタイミングを指定します。詳細は，マニュアル「OpenTP1 システム定義」を参照してください。

また，scdls コマンドの-p オプションによって，スケジューラデーモンごとの処理スレッドの利用状況を表示できます。

処理スレッドが一時的に不足するシステム例を次の図に示します。

図 C-7　処理スレッドが一時的に不足するシステム例



付録 C.3　マルチスケジューラ機能を使用したシステム構成例

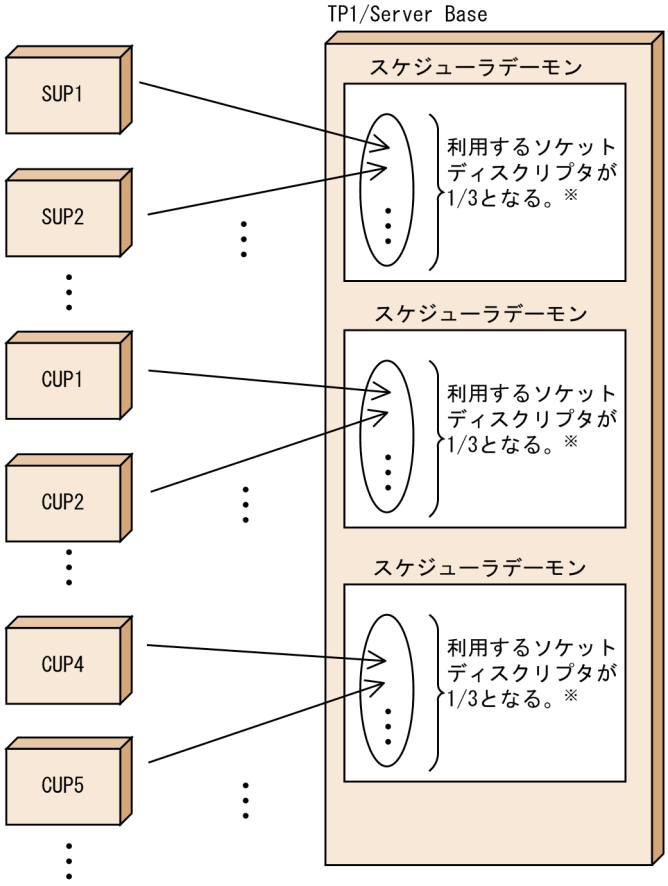
マルチスケジューラ機能を使用すると，スケジューラが原因となるおそれのある問題を解決できます。ここでは，マルチスケジューラ機能を使用したシステム構成について説明します。

(1)　ソケットディスクリプタの不足を解決するシステム構成

スケジューラデーモンに接続するクライアント UAP を分散することで，1 スケジューラデーモンが利用するソケットディスクリプタ数を少なくできます。

ソケットディスクリプタの不足を解決するシステム構成例を次の図に示します。

図 C-8　ソケットディスクリプタの不足を解決するシステム構成例



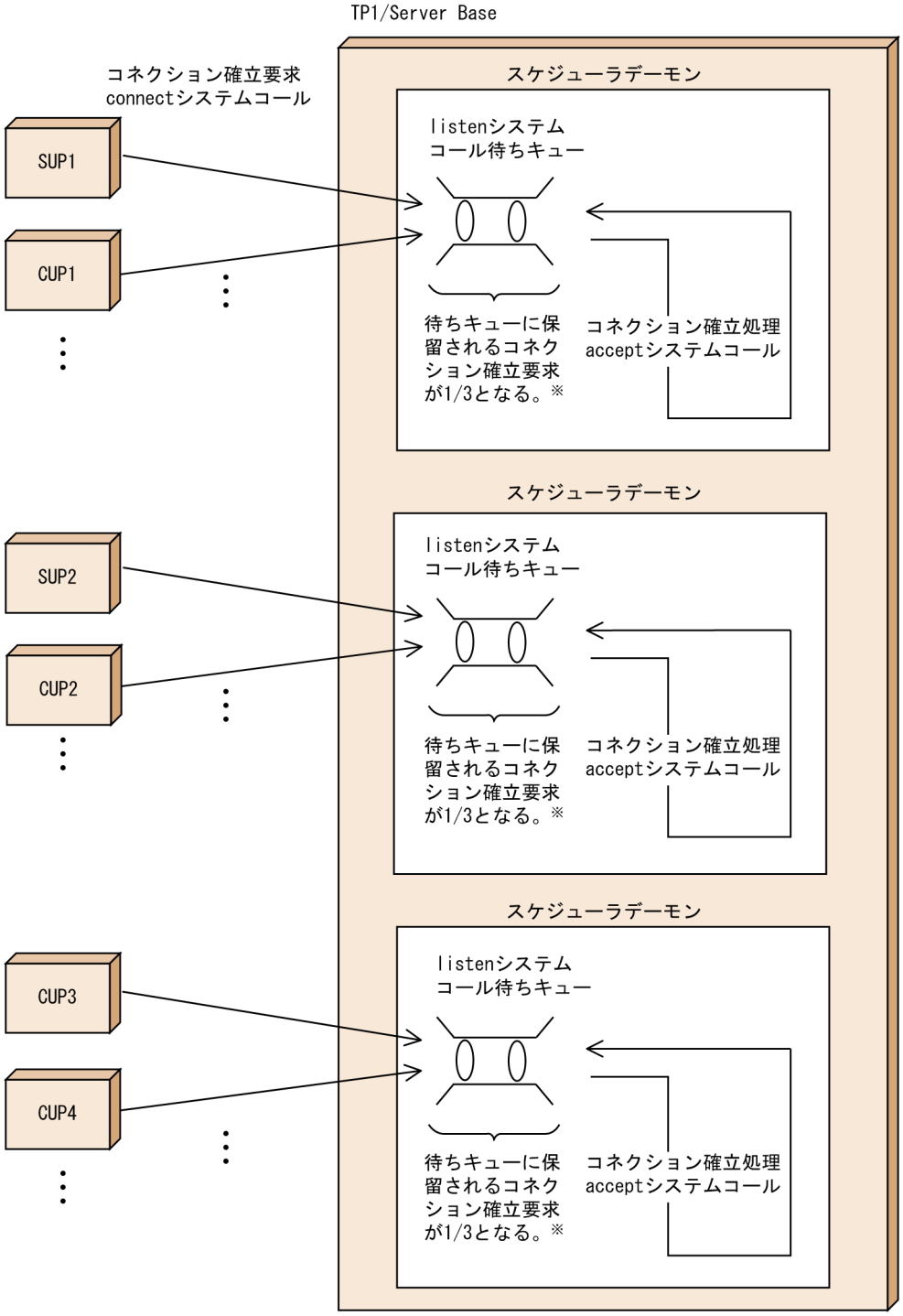
注※　スケジューラデーモン数を3と指定した場合です。

(2)　connect システムコールのエラーを解決するシステム構成

スケジューラデーモンに接続するクライアント UAP を分散することによって，listen システムコールの待ちキューとして保留されるコネクション確立要求数を少なくできます。

connect システムコールのエラーを解決するシステム構成例を次の図に示します。

図 C-9　connect システムコールのエラーを解決するシステム構成例



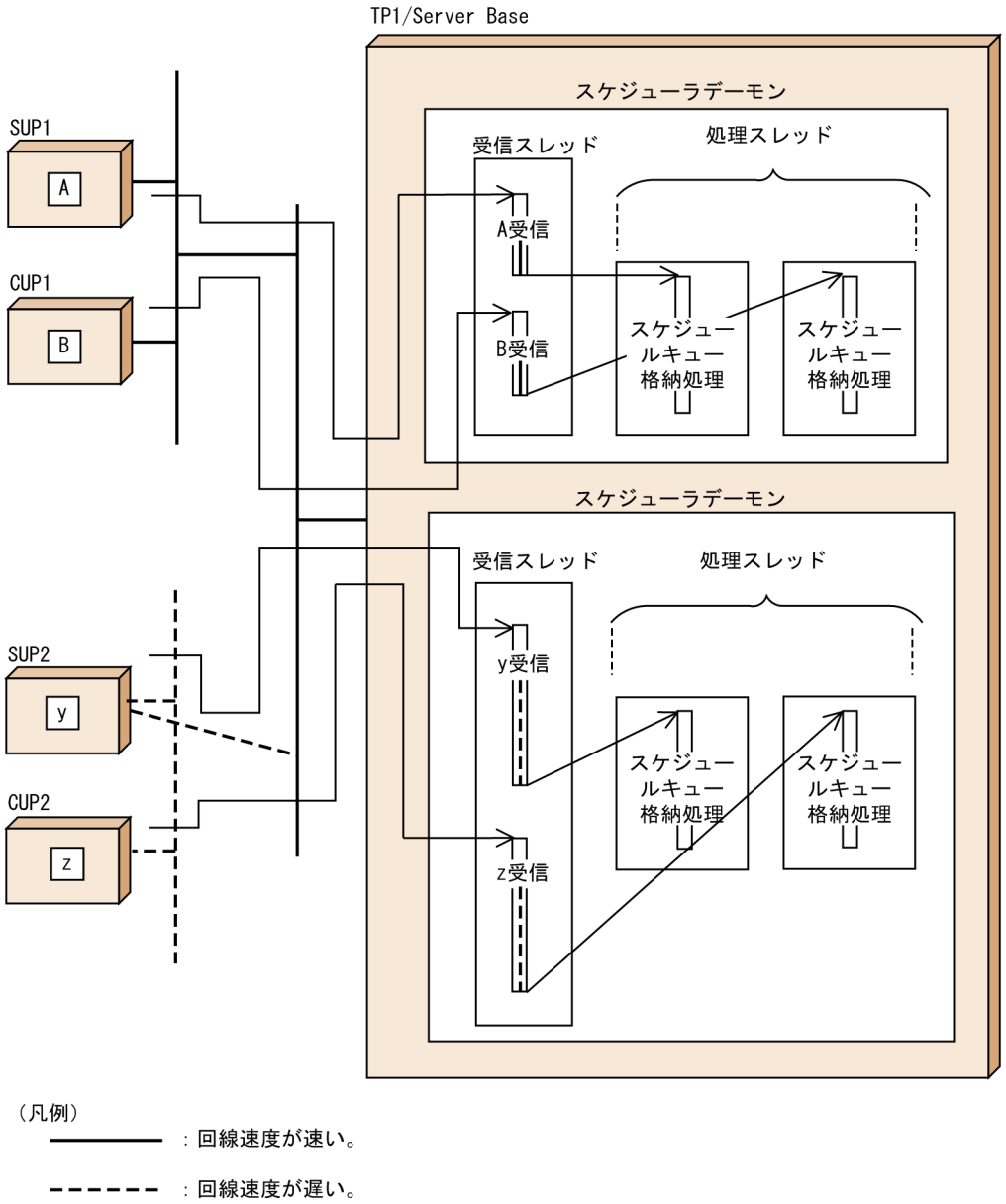
注※　スケジューラデーモンを3と指定した場合です。

(3)　回線速度が速いネットワークを有効利用できるシステム（回線速度が異なるネットワークを混在して利用している場合）

回線速度が速いネットワークのクライアント UAP を処理するスケジューラデーモンと，回線速度が遅いネットワークのクライアント UAP を処理するスケジューラデーモンを分けることで，回線速度が速いネットワークを有効に利用できます。

回線速度が速いネットワークを有効利用できるシステム例を次の図に示します。

図 C-10　回線速度が速いネットワークを有効利用できるシステム例

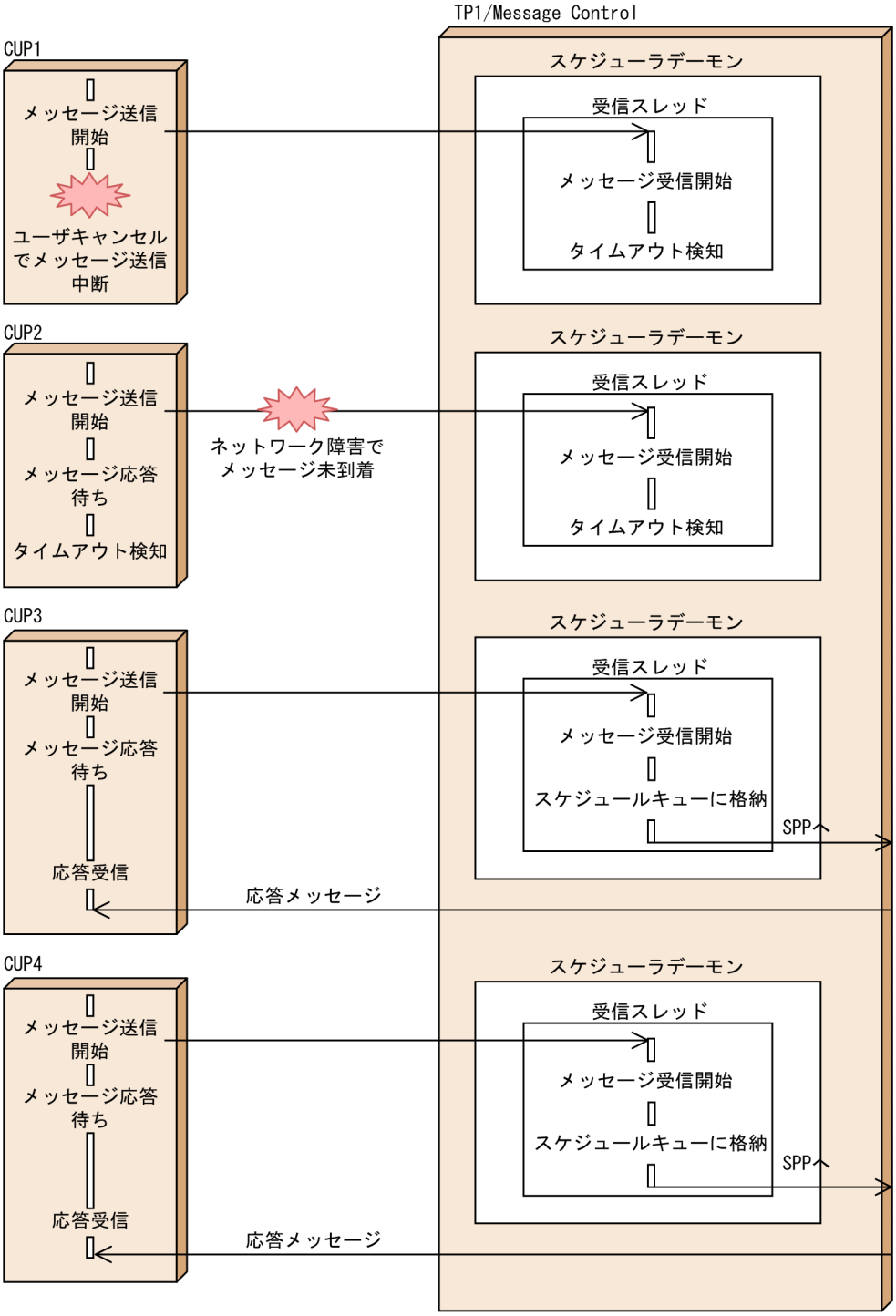


(4)　サービス要求メッセージが途切れないシステム

マルチスケジューラ機能を用いて，メッセージ受信処理が滞るスケジューラデーモンを局所化すると，ほかのサービス要求メッセージを滞らせることなくスケジューリングできます。

サービス要求メッセージが途切れないシステム例を次の図に示します。

図 C-11　サービス要求メッセージが途切れないシステム例

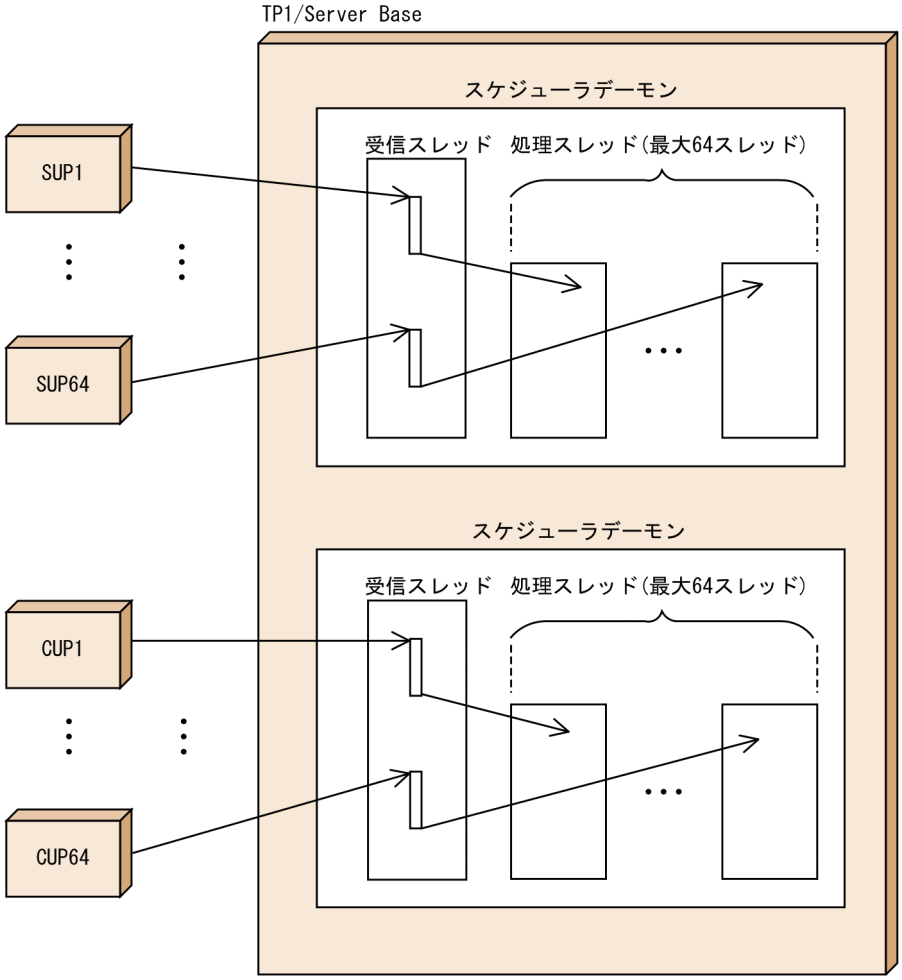


(5)　同時実行可能な処理スレッド数を増やしたシステム

マルチスケジューラ機能を用いて，同時に実行できる処理スレッドの数を増やすことができます。これによって，処理スレッド不足による一時的な通信障害やタイムアウトを回避できます。

同時実行可能な処理スレッド数を増やしたシステム例を次の図に示します。

図 C-12　同時実行可能な処理スレッド数を増やしたシステム例



付録 C.4　注意事項

1. マルチスケジューラ機能は TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証されません。

   マルチスケジューラ機能の詳細は，マニュアル「OpenTP1 システム定義」または「OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」を参照してください。

2. マルチスケジューラ機能を適用する場合には，システム定義の変更に加えて，スケジューラデーモンの増加に伴うカーネルパラメータの変更が必要となる場合があります。

3. OpenTP1 システム内の同一サービスグループに，マルチスケジューラ機能を使用しているユーザサーバと，マルチスケジューラ機能を使用していないユーザサーバが混在する場合，次のことに注意してください。

   - マルチスケジューラ機能を使用したサービス要求は，マルチスケジューラ機能を使用しているユーザサーバに優先して負荷分散されます。
   - マルチスケジューラ機能を使用したサービス要求は，マルチスケジューラ機能を使用しているユーザサーバが高負荷状態でも，マルチスケジューラ機能を使用していないユーザサーバには負荷分散

されません。マルチスケジューラ機能を使用しているユーザサーバが高負荷状態の場合に，マルチスケジューラ機能を使用していないユーザサーバに負荷分散するには，スケジュールサービス定義の scdmulti 定義コマンドに-t オプションを指定してください。scdmulti 定義コマンドの詳細については，マニュアル「OpenTP1 システム定義」のスケジュールサービス定義を参照してください。

索引

D

E

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

に

ね

の

は

ひ

ふ

へ

ほ

ま

み

め

ゆ

ら

り

れ

ろ

わ