UMLからSystemCへのモデリング（DMAコントローラ）の事例

株式会社リコー
画像システム事業本部　プラットフォーム開発センター
第二開発室　開発１グループ
主席係長 宮原　忠義・主任 難波　睦

近年システムＬＳＩの設計も複雑化してきています。ＳｏＣ開発も上流設計の必要性が迫られてきました。ここでは、ＵＭＬ（Unified Modeling Language）よりDMAコントローラをモデリングした事例を紹介いたします。

【USoCF参加に至るまでの背景】

筆者が所属する部署では、コピー、プリンタ、ＦＡＸ等の複合機における数百万ゲート規模のコントローラＡＳＩＣ及び画像処理ＡＳＩＣを開発しています。

本部署内において、ＡＳＩＣ開発プロセスの改革を目指したワーキンググループ（後述WGと記す）を発足しました。本WGでは、上流設計、ＳｙｓｔｅｍＣ、ＵＭＬ、オブジェクト指向、協調設計、協調検証、仮想ハード等の様々な技術に目を付け、導入を検討することに日夜励んでいます。まずは、ＵＭＬに着目し、試行してみようと言うことで、ＤＭＡ（Direct Memory Access）コントローラの仕様書を分析し、ＵＭＬ図化を実施しました。

また、次世代開発言語として話題のＳｙｓｔｅｍＣをとりあげ、WG内で調査を始め、抽象度をあげれば今まで遅くて避けていたハードウェアとソフトウェアの協調シミュレーションが早くなることが望まれ、実用化できるのではないかと考え、ＡＳＩＣの仕様書を分析し、仕様の抽象度を上げて、ＳｙｓｔｅｍＣにて記述しました。そんな中、ＵＳｏＣＦ（UML for SoC フォーラム）の存在を知り、参加する事となりました。以下実際に行ったモデリングの事例をご紹介いたします。

【実証プロジェクトの題材は　ＤＭＡコントローラ】

今回の実証プロジェクトの題材はDMAコントローラとしました。前述にもあるように社内WGで試行していたからです。DMAコントローラは、コントローラＡＳＩＣの要であることと、仕様が理解しやすく、社内やＵＳｏＣＦ等で発表した場合、誰もがとっつきやすいと思ったのが選択の理由です。

【実証プロジェクトのゴール】

前述の通り、筆者はＵＳｏＣＦの活動であるＵＭＬ拡張プロファイルの実証を行いました。実証のゴールはＳｙｓｔｅｍＣソースコードスケルトン作成までとしました。詳細コーディングしてシミュレーションまでを目指したかったのですが、本来の業務の合間に行うので実証にさける時間がとれないこととスケルトンまでたどり着ければ実証することと同等であろうと判断し、スケルトンに決定しました。もし、詳細をコーディングして動かなかったとしたら、ＵＭＬ拡張プロファイルが原因ではなく、モデリングが悪いからであると思っています。ということで、ここで記す事例は、動作確認していないことをご了承ください。

【要求仕様（要求仕様書作成）】

要求仕様は、現在設計しているＤＭＡコントローラ仕様書から仕様を抽出し、納期達成範囲内でできるものであり、且つ、一般公開可能なものを抜粋しました。それが表１の通りです。また図１にＤＭＡコントローラのデータ転送イメージを示します。要求仕様にある「２次元データ」とはデータを２次元で管理していることであり、画像をイメージして幅とライン数を設定し、これを転送の単位とするものです。（図２）また、要求仕様にある「ディスクリプタ」というのは、ひとつの転送に必要な情報がメモリに複数格納してあり、ある転送が終わったら次のディスクリプタ情報を入手し、また転送を始めるという様に転送を数珠繋ぎに行うものです。（図３）

なお、以降、上記ひとつの転送のことを１バンドの転送と記述します。

【開発プロセス】

本実証プロジェクトの開発プロセスは、図４のように反復型開発プロセスにて実施しました。

ＭＤＡ（Model-Driven Architecture）等実施している先進的なソフトウェア開発者にとっては当たり前かもしれませんが、ＡＳＩＣ開発では現在でもウォーターフォール型が主流です。筆者にとっては慣れていない分、新鮮な気分を感じつつ開発を実施できました。後戻りしながらモデルが洗練されていくことに感動すら覚えました。なにせウォーターフォールにおいては後戻りは許されないのですから・・・。

UMLからSystemC、そして高位合成によりＡＳＩＣが開発できる世界になれば（もうできているかもしれませんが)、反復型開発プロセス＋モデリングが今後当たり前になるのでしょう。なお、本プロセスは筆者らチーム独自のプロセスであり、これをＵＳｏＣＦが提唱しているわけではありません。

【要件定義工程（ユースケース図)】

この工程では、開発対象となるシステムについて、実現すべき明確なゴールの設定、およびゴール達成に必要な業務領域（ドメイン）の範囲の明確化と問題の理解、システムに要求される機能の洗い出しを行います。本事例ではシステムはＤＭＡＣに該当します。簡単に言ってしまうと、「ＤＭＡＣが何をするのか？」ということを明確にするのです。本工程での成果物は、図５のユースケースと図６のユースケース記述です。ユースケースにより、対象システムのおおよその規模を推測することができます。ＤＭＡＣのユースケースは「データを転送する」とシンプルに定義しました。

【分析工程（クラス図、コラボレーション図)】

この工程では、対象システムに一歩踏み込んで、前工程で抽出したユースケース実現のために何が必要かを検討します。具体的にはクラスの抽出とクラス間の協調関係をUMLのクラス図やコラボレーション図を使って模索します。慣れない内はこの段階で実装手段に目が行きがちですが、それは後々設計の自由度を狭めてしまいます。ＤＭＡＣの例では一旦「ハードウェアで実装する」ことや「ハードウェアであることによる制約」は忘れます。ソフトウェアとの通信は通常レジスタを使用しますが、ここでは直接通信できることとして扱います。メモリデバイスとの通信も、バースト長などの制約はありますが、同じようにいかなる大きさのデータも瞬時に転送できることとします。通信を理想化することにより機能のエッセンスを浮き彫りにすることがここでのポイントです。

図７に本工程の成果物である分析モデルを示します。

【設計工程１（クラス図)】

この工程では実装に向かって前工程の分析モデルを詳細化してゆきます。DMACの例では、対メモリおよび対画像処理ユニットの通信制約を採り入れました。それぞれに対して一度にアクセスできるデータ量は決まっているので、バッファを使って「バケツリレー」でデータを転送することとします。

一方でソフトウェアとの通信はモデル化を敢えて避けています。レジスタアクセスとレジスタそのものをクラス図に導入しようとすると煩雑になってしまうのと、レジスタの実装方法がほぼパターン化しているのでモデルの必要性が小さいからです

図８に本工程の成果物である設計モデルを示します。

なお、ここではまだＵＳｏＣＦ提唱のUML拡張プロファイルは導入していません。

【設計工程２（拡張プロファイル)】

この工程でいよいよＵＭＬ拡張プロファイルを導入しコーディングを行います。その際、拡張プロファイルを考慮し英語化しました。図９に本工程の成果物であるクラス図を示します。

ZIPC WATCHERS Vol.8

このクラス図は、ソフトウェアで使われているクラス図とはかなり異なっています。特徴的な点としては、属性にあるポート（<<SoCPort>>）が全てパブリックになっています。これは、ＳｙｓｔｅｍＣ言語の特徴の一つで各モジュール間の通信はポートを通して行われます。そのポートがモジュールの属性として宣言される為です。つまり、図８にあるクラス中のメソッドから必要なポートの抽出を行う必要が出てきます。ここで注意する点は、単純に図８のメソッドと図９のポートは１対１に対応するものではありません。例えば、図８の転送管理クラスにあるメソッド「起動」「停止」は、全て1Bitの情報（bool型）で伝えることが可能ですが、図９のクラス図ではそれらを一つにまとめて「param_info」というポートにしています。これは、今後ＳｙｓｔｅｍＣでコーディングすることを考えると、１つのポートにして通信データをパッケージ化した方が変更する場合に有利な為です。また、構造図にしたときを考えるとモジュール間のチャネルも少ない方が見やすいというメリットもあります。

もう一つの特徴としては、図９のクラス図中のメソッド（<<SoCProcess>>）です。これは、図８のクラス図には存在しません。これもポートと同じくＳｙｓｔｅｍＣ言語の特徴になります。このプロセスの概念については誌面の都合上省略します。このように、図８のクラス図と図９のクラス図とでは大きなギャップがあり、今後検討するべき課題も多く残っています。

【構造図】

構造図では拡張プロファイルを使って定義したモジュール群の静的な構造を表現します。ポートやチャネルが明示され、通信の流れを理解することができます。ただし、構造図からは個々のモジュールが持つ属性やプロセスは見えないため、前工程の拡張プロファイルを導入したクラス図とつき合わせることでモデルの全体像を把握することができます。

本構造図は、キャッツ製「XModelink」を使用して作成しました。

まずはじめに、図９より作成したモデルよりクラス図インポート機能を使ってクラスを読み込みます。その際、個々のモジュールが持つ属性やプロセスも一緒に読み込まれます。そしてお絵かきツール的感覚で、SystemC表記法によりポート、チャネル、接続などを追加します。ここで作成した構造図を図10に示します。

本来ここで詳細記述を行いたい所ですが、今回は省略しました。

【SystemCスケルトン自動生成とコードチェック】

本実証プロジェクトのゴールであるSystemCスケルトンは、「XModelink」上で作成した構造図により、自動生成しました。（図11）

次に「XModelink」上にて、本SystemCコードを正常にコンパイル出来るかどうかの確認を行いました。

【XModelinkツールを使った方法】

上記の如く、今回キャッツ製「XModelink」を使用して、ＵＭＬ→SystemC構造図→SystemCスケルトン→スケルトンチェックまでの作業を行いました。

今回は５つ程度のクラスでしたが、複雑になればなるほど作業効率向上の効果を期待できると思います。

また、SystemC初心者であってもコンパイル可能なソースコードを容易に作成可能であると思われます。

UMLモデルとの連携、SystemC表記法に基づく構造図、コンパイル可能ソースコード自動生成まで対応しているツールは他には類はなく、ハードウェア界のMDA到来を予感します。

参　考

【USoCFとは】

ＵＳｏＣＦとは、正式名「UML for SoC フォーラム（USoCF）」と呼ばれる団体であり、ソフトウエアの世界では標準とされているＵＭＬにより複雑な上流設計の適用がＳｏＣにも有効である！と考え設立されました。主な活動内容は、拡張プロファイルをOMGに提案、DAC2004のWorkShopにて発表、今回紹介したUML提要事例など...。

【XModelink SoCModelerとは】

SystemC表記法によりグラフィカルに設計が行え、SystemCコードを自動生成します。また、ＵＭＬよりモデリングしたクラス図をインポートする機能などがあります。ＵＭＬ→ＳｏＣへの取り組みが注目され、「2004年、ESEC LSIオブザイヤー　グランプリ」を受賞しました。ＳｏＣ業界でも今後上流設計が必要とされる事が予想され、先駆けて取り組んでいる事が評価されました。

【拡張プロファイルとは】

本来UMLはソフトウエアのための記述を行うものであり、システムＬＳＩを記述するためのものとして開発されたものではありません。そこでUSoCFとしては、ＵＭＬにてシステムＬＳＩを記述するためのプロファイルとして拡張プロファイルを作成しました。

またさらに標準化を目指してＵＭＬのＳｏＣ向け拡張プロファイルを作成してＯＭＧに提案を行っています。

著者プロフィール

■宮原忠義

'86年にリコーに入社し、電源、モータ等のアナログ設計を経て、
'95年よりASICの回路設計及び、検証に従事。
'02年よりASICプロセスの改革を目指し、ワーキンググループ的活動で上流設計等の検討を始め、現在に至る。

■岡田　敏

'98年にリコーに中途入社し、組み込みソフト業務に従事する。ここで、構造化分析からオブジェクト指向分析へのパラダイムシフトを体験。
'01年末より、ＡＳＩＣ（主に画像処理系）の論理設計、検証に従事。
'02年より、ワーキンググループ活動を通じてASICプロセスのパラダイムシフトを体験中。

■難波　睦

'98年リコー入社以来、ASICの論理設計、検証に従事。テストベンチのC++化をきっかけにオブジェクト指向に触れ、以後UML等を使ったASIC上流設計の検討活動をおこなっている。

表１　DMAC（DAMコントローラ）に関する要求仕様

No	要求仕様
1	画像処理ユニットとメモリ間でデータ転送を行う。
2	データ転送方法は、ディスクリプタ情報に従う
3	転送するデータは、2次元データのみの転送
4	データ転送終了後は、割り込み通知を行う。
5	メモリ領域外にアクセスした場合は、エラー割り込み通知を行う。

図１　画像処理ユニットとメモリ間でのデータ転送イメージ図

メモリから画像処理ユニットへのデータ転送は、DMACがメモリからデータを読み込み、画像処理ユニットに対してデータ転送を行います。
画像処理からメモリへのデータ転送は、画像処理ユニットがDMACへデータを転送し、DMACがメモリに対してデータを書き込みます。

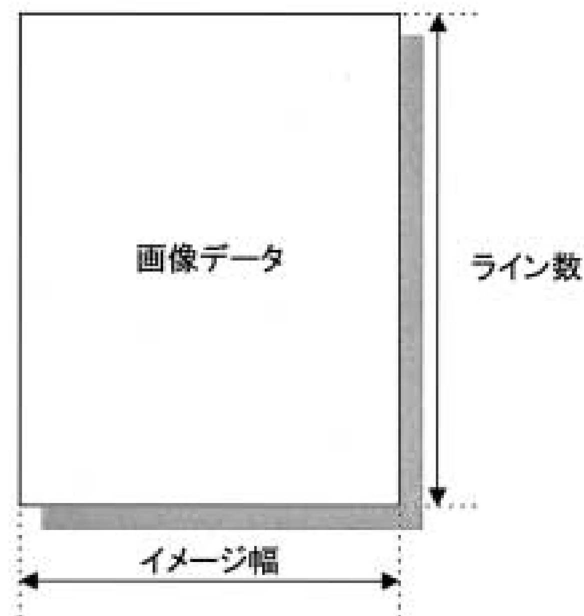

図２　２次元管理のイメージ図
メモリ上に画像データを展開した時のイメージ図です。
画像データの横側をイメージ幅とします。単位は、画素数です。
画像データの縦側をライン数とします。単位は、ライン数です。

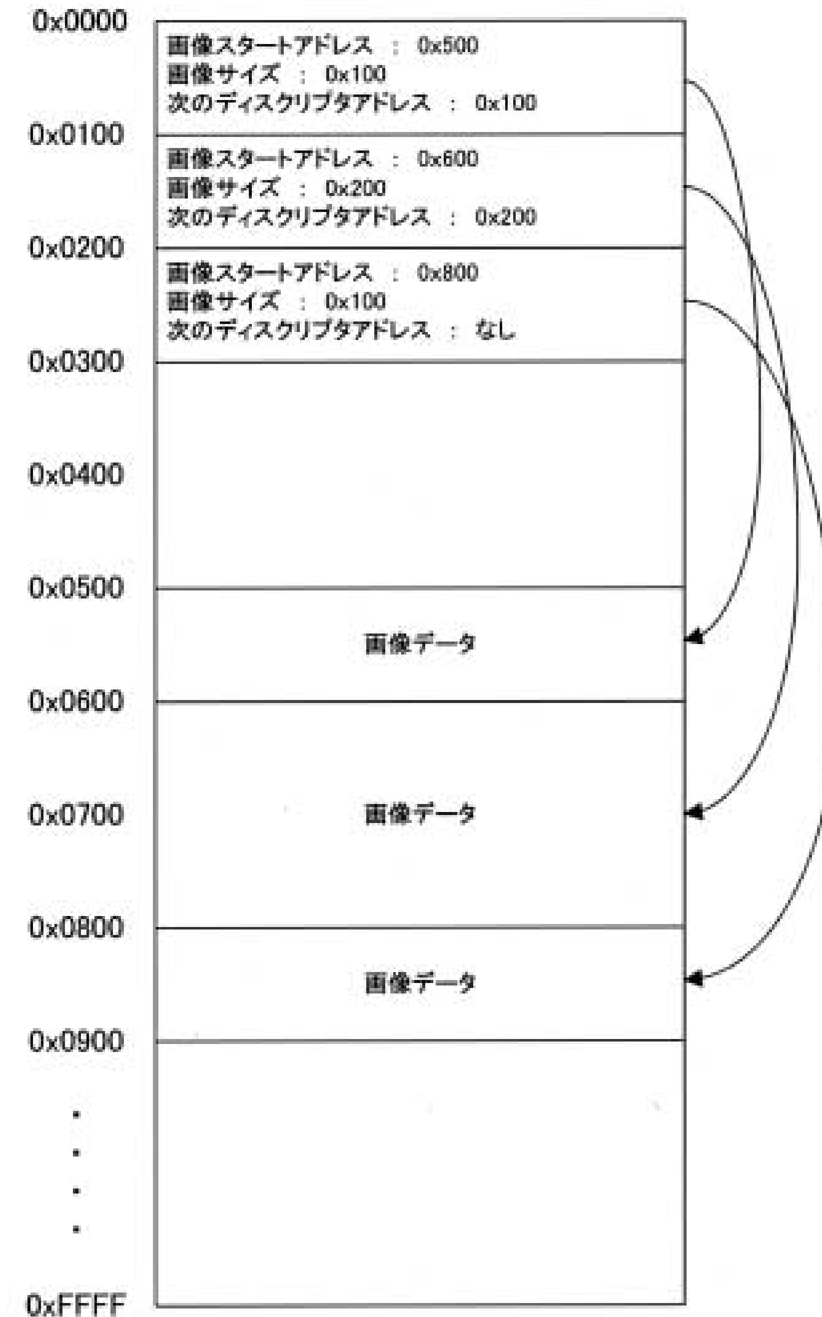

図３　ディスクリプタ方式のイメージ図
左にある数値0x0000～0xFFFFは、メモリ上のアドレスを示しています。
例えば、アドレス0x0000～0x0100に最初のディスクリプタ情報があります。
最初のディスクリプタの中に画像データのスタートアドレスがあるのでDMACは、そのアドレスから画像データを画像サイズ分読み出します。（その他にも情報がありますが、ここでは省略しています。）そして、次のディスクリプタアドレスからディスクリプタ情報を取得して・・・、という具合に画像データが終了するまで繰り返します。

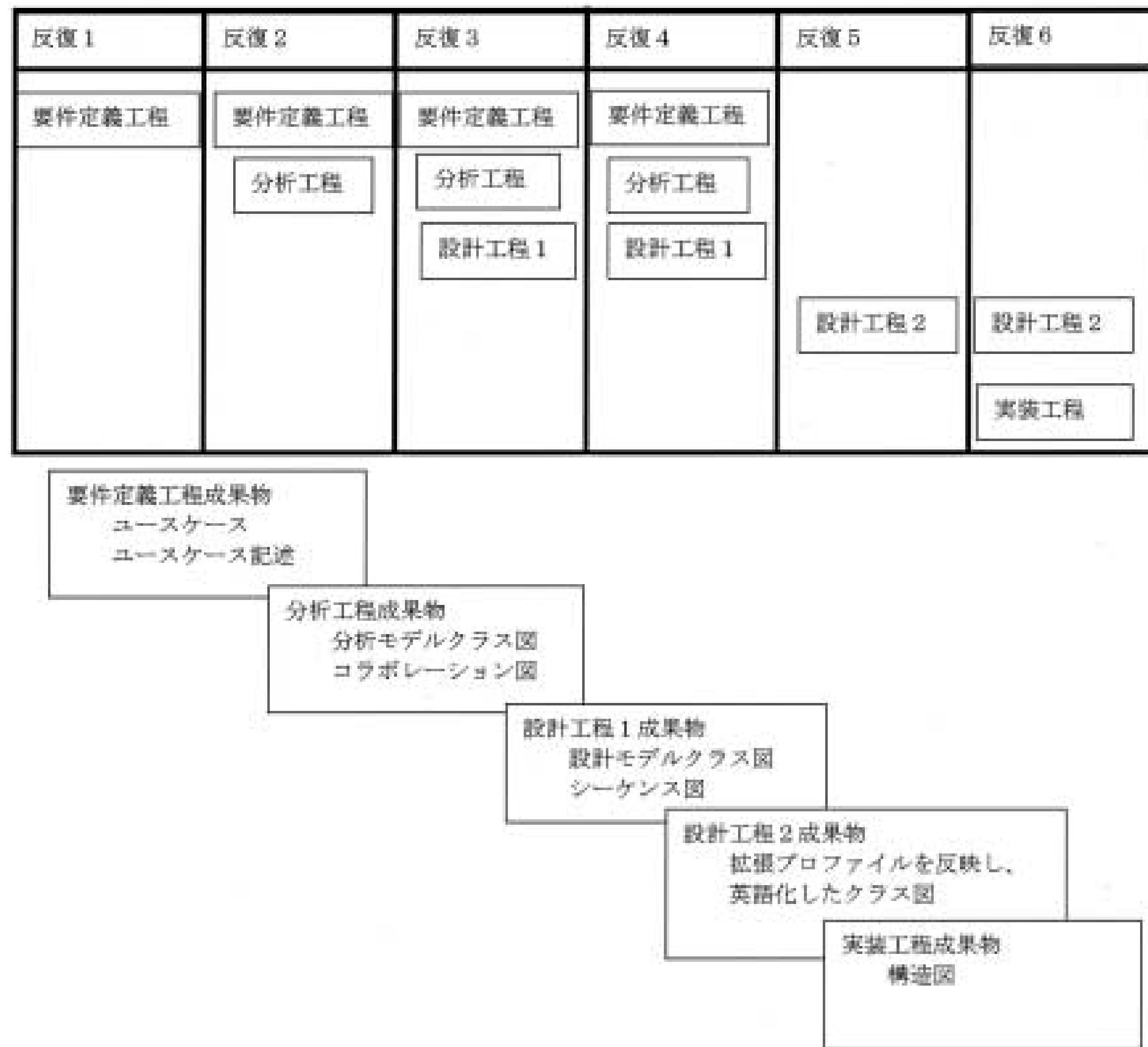

図４．反復型開発プロセス
要件定義工程、分析工程、設計工程１、設計工程２、実装工程を繰り返すことにより、各工程の成果物を洗練した。

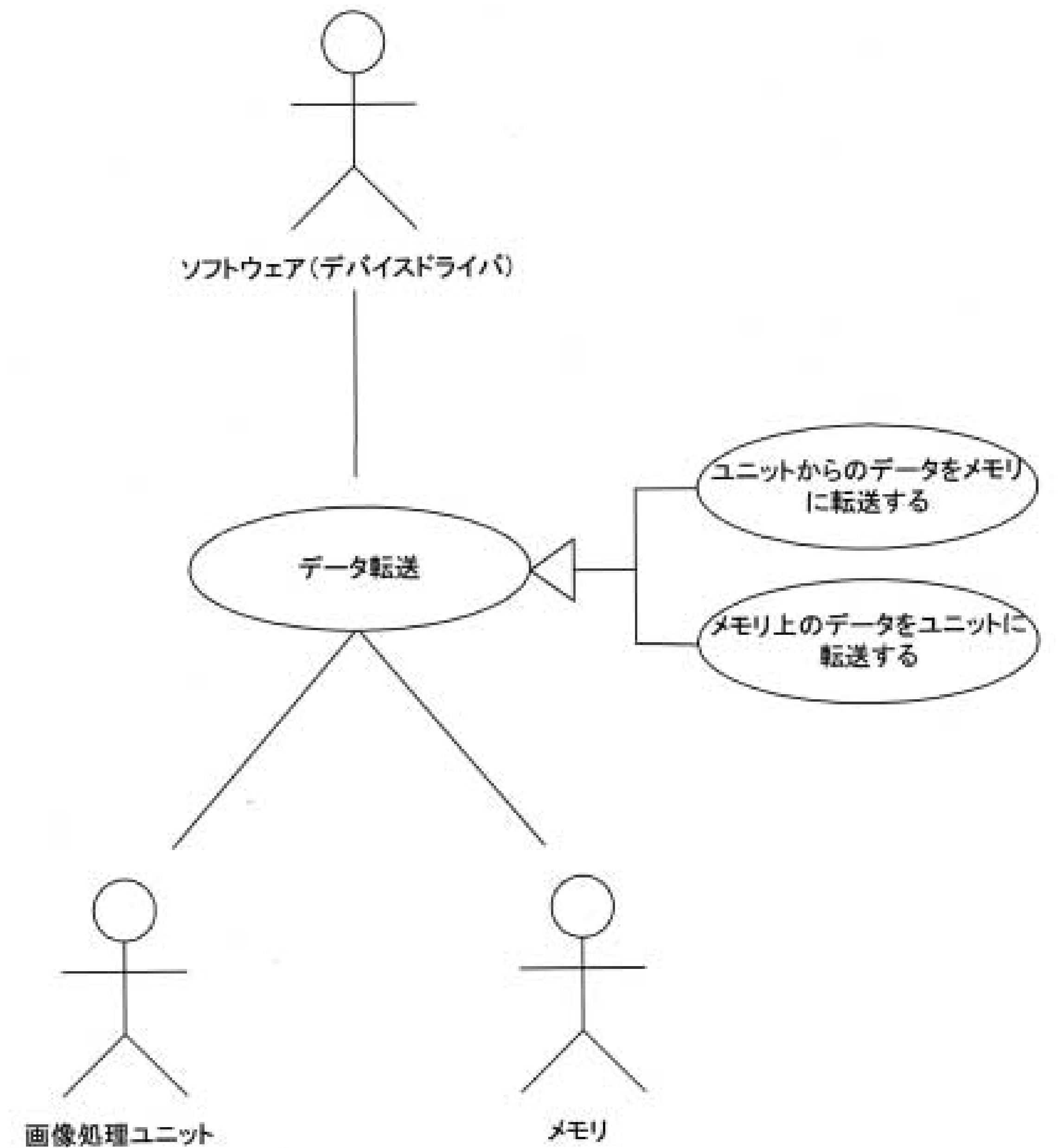

図５ UMLユースケース図
DMACの仕事とステークホルダーを明らかにする。DMACはソフトウェアの指示により画像処理ユニットとメモリ間でデータ転送をおこなう。転送の方向が２種類ある。

ユースケースID	UC001
概要	画像処理ユニットからのデータをメモリに転送する
アクター	１．ソフトウェア（デバイスドライバ）
	２．画像処理ユニット
	３．メモリ
事前条件	・画像処理ユニット側のデータ転送準備ができていること
	・ディスクリプタがメモリ上に配置されていること
事後条件	・データがメモリ上に転送されていること
基本系列	１．アクター１はDMACに対してディスクリプタのアドレスを設定する
	２．アクター１はDMACに対して画像のイメージ幅を設定する
	３．アクター１はDMACに対して画像のメモリ幅を設定する
	４．アクター１はDMACに対して転送開始を指示する
	５．DMACはディスクリプタを元にアクター２からデータを受け取りアクター３
	に転送する
	６．DMACは全ての転送が完了したらアクター１に完了を通知する
代替系列	
例外系列	・データ転送中、アクターはDMACの強制停止ができる
	・メモリアクセスエラー（アクターにエラー通知）
備考	・DMACからアクターへの通知は割り込みによりおこなう。
参考文献	

ユースケースID	UC002
概要	メモリ上のデータを画像処理ユニットに転送する
アクター	１．　ソフトウェア（デバイスドライバ）
	２．　画像処理ユニット
	３．　メモリ
事前条件	・画像処理ユニット側のデータ受け取り準備ができていること
	・ディスクリプタがメモリ上に配置されていること
	・データがメモリ上に配置されていること
事後条件	・データが画像処理ユニット側に転送されていること
基本系列	１．アクター１はDMACに対してディスクリプタのアドレスを設定する
	２．アクター１はDMACに対して画像のイメージ幅を設定する
	３．アクター１はDMACに対して画像のメモリ幅を設定する
	４．アクター１はDMACに対して転送開始を指示する
	５．DMACはディスクリプタを元にアクター３からデータを読み出しアクター２
	に転送する
	６．DMACは全ての転送が完了したらアクター１に完了を通知する
代替系列	
例外系列	・データ転送中、アクターはDMACの強制停止ができる
	・メモリアクセスエラー（アクターにエラー通知）
備考	・DMACからアクターへの通知は割り込みによりおこなう。
参考文献	

図６　ユースケース記述

ユースケース記述は各ユースケースをより詳細に文書化したものである。ユースケース記述そのものはUMLではない。ユースケース記述を書くことで、より分析を深堀する。

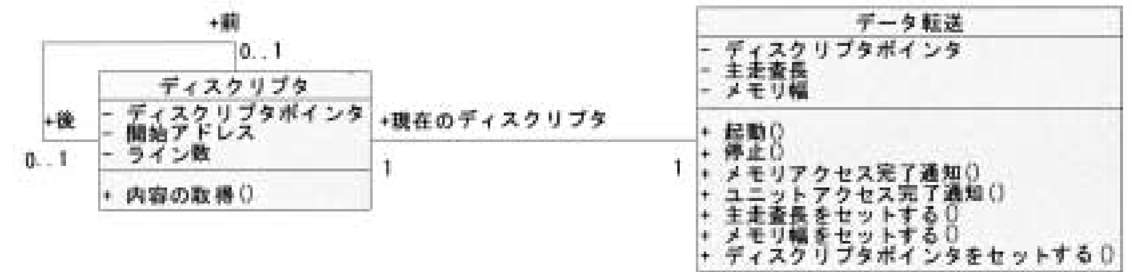

図７　UML分析クラス図

DMACのデータ転送とディスクリプタの関係をモデル化する。実装イメージをできる限り排除して、「何を設計するのか」を明らかにするのが分析モデルのポイント

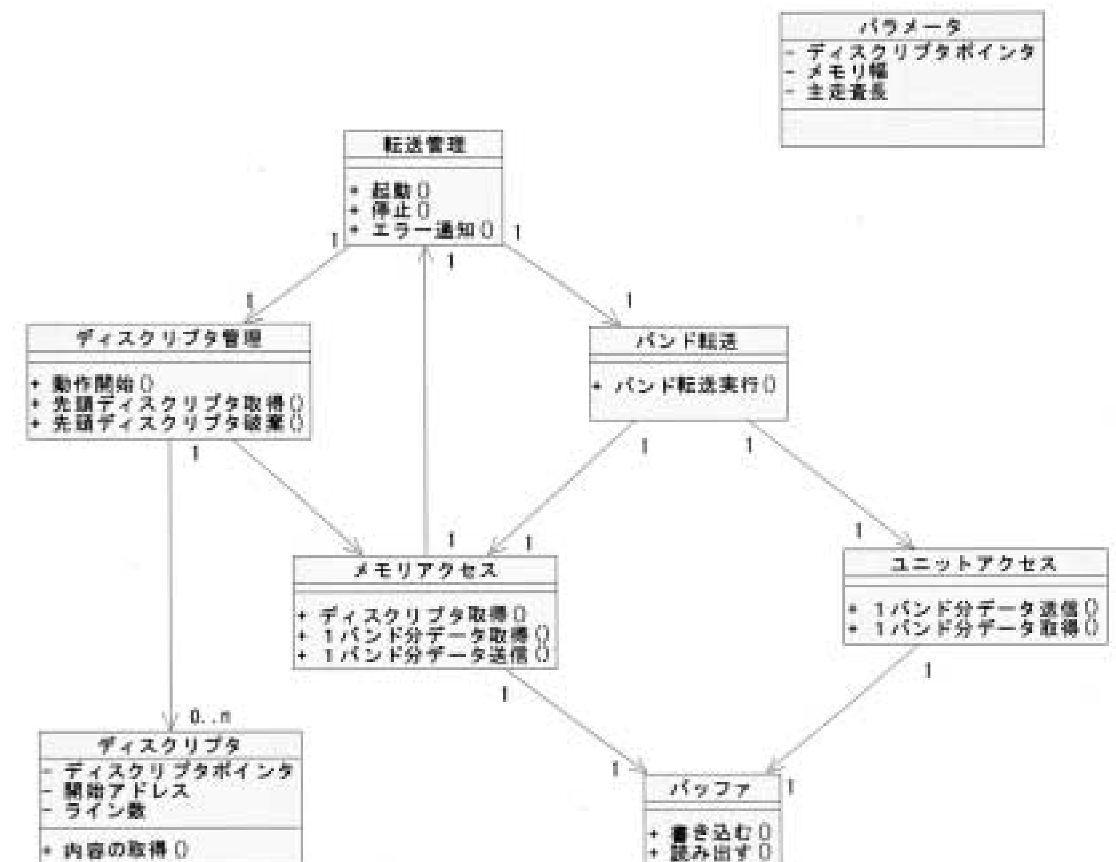

図８　UML設計クラス図

対メモリ、対ユニットのデータ転送の制約等を反映したクラス図。ハードの内部レジスタについては敢えてモデル化せず、ソフトがあたかも直接ハードのメソッドを呼び出せるかのようにモデリングするにとどめる。モデルの複雑化防止のためである。

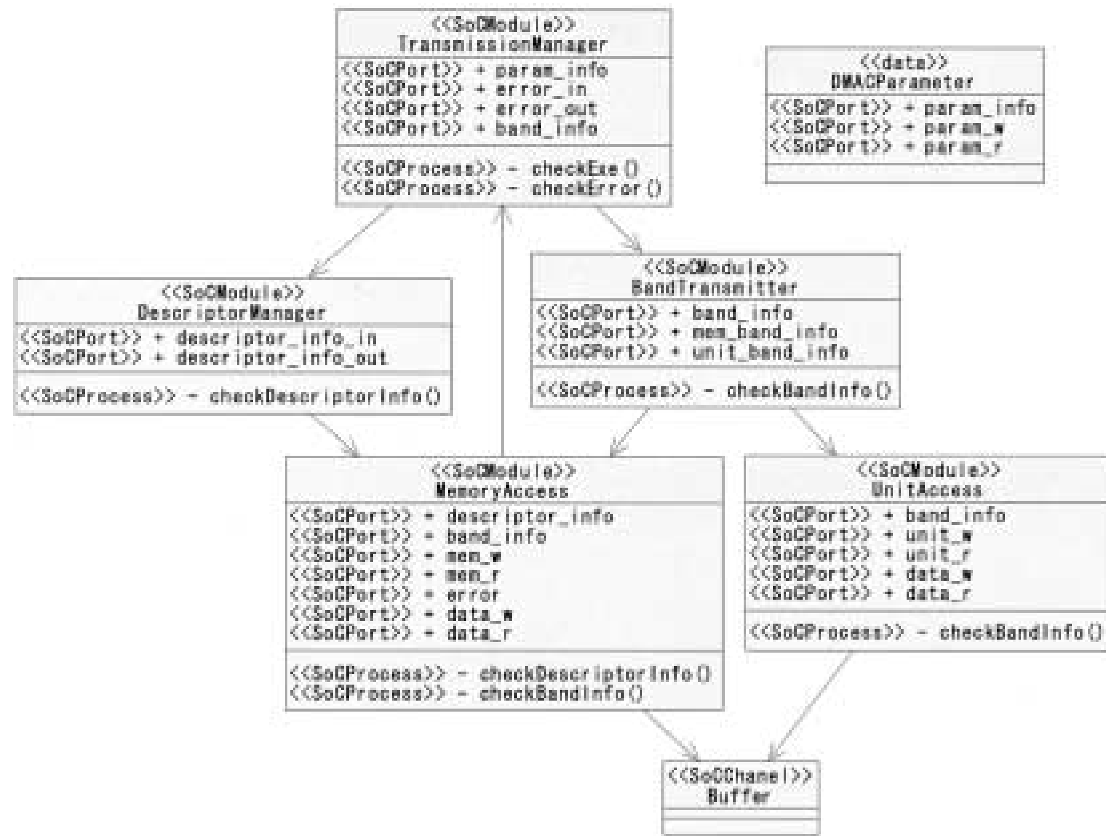

図９　拡張プロファイルを反映したクラス図

ハードウェア化を意識したクラス図です。
また、ここではUSoＣＦで検討された拡張プロファイルを使っています。
（多重度は、全てが１対１だったのでここでは省略しています。）

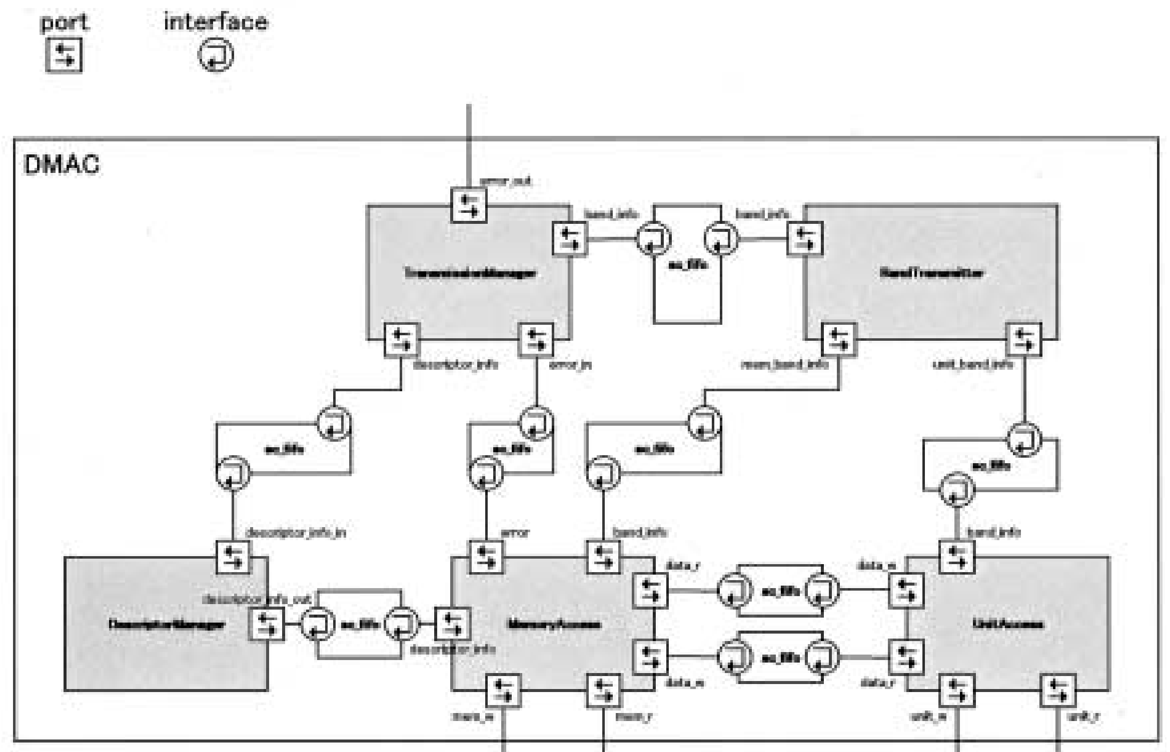

図10　構造図

図９で作成した、クラス図を元に構造図（インスタンス図）を作成しました。
クラス間の関連にデータ転送の為に必要なチャネルを挿入しています。

図11

XModelinkより自動生成したＳｙｓｔｅｍＣコード。ポート接続、モジュールヘッダーファイルなど...。