TOPPERS 活用アイデア・アプリケーション開発コンテスト

部門　　：(活用アイデア部門)
　　アプリケーション開発部門

作品のタイトル　：TECS と PlantUML の連携

作成者　　：長島 宏明

共同作業者　　：

対象者　　：TECS CDL を記述する開発者

使用する開発成果物　　：TECS

目的・狙い

TECS を使った開発を手軽な物とするため、UML のコンポーネント図と関連させ、より一般的な表現で CDL の記述を支援するツールを提案する。
多くの解説の見つかる UML と関連付けることで、TECS の理解を助け、TECS の導入を容易にする。

アイデア/アプリケーションの概要

PlantUML のコードと連携し CDL との相互変換するツールを作ることで、UML のコンポーネント図を元にした、CDL のコーディングを支援する。
また、CDL の記述から PlantUML のコードを出力し、UML のコンポーネント図を作成することで、設計資料などで使え、コードと設計書の並行開発が行えるような環境の提案をする。

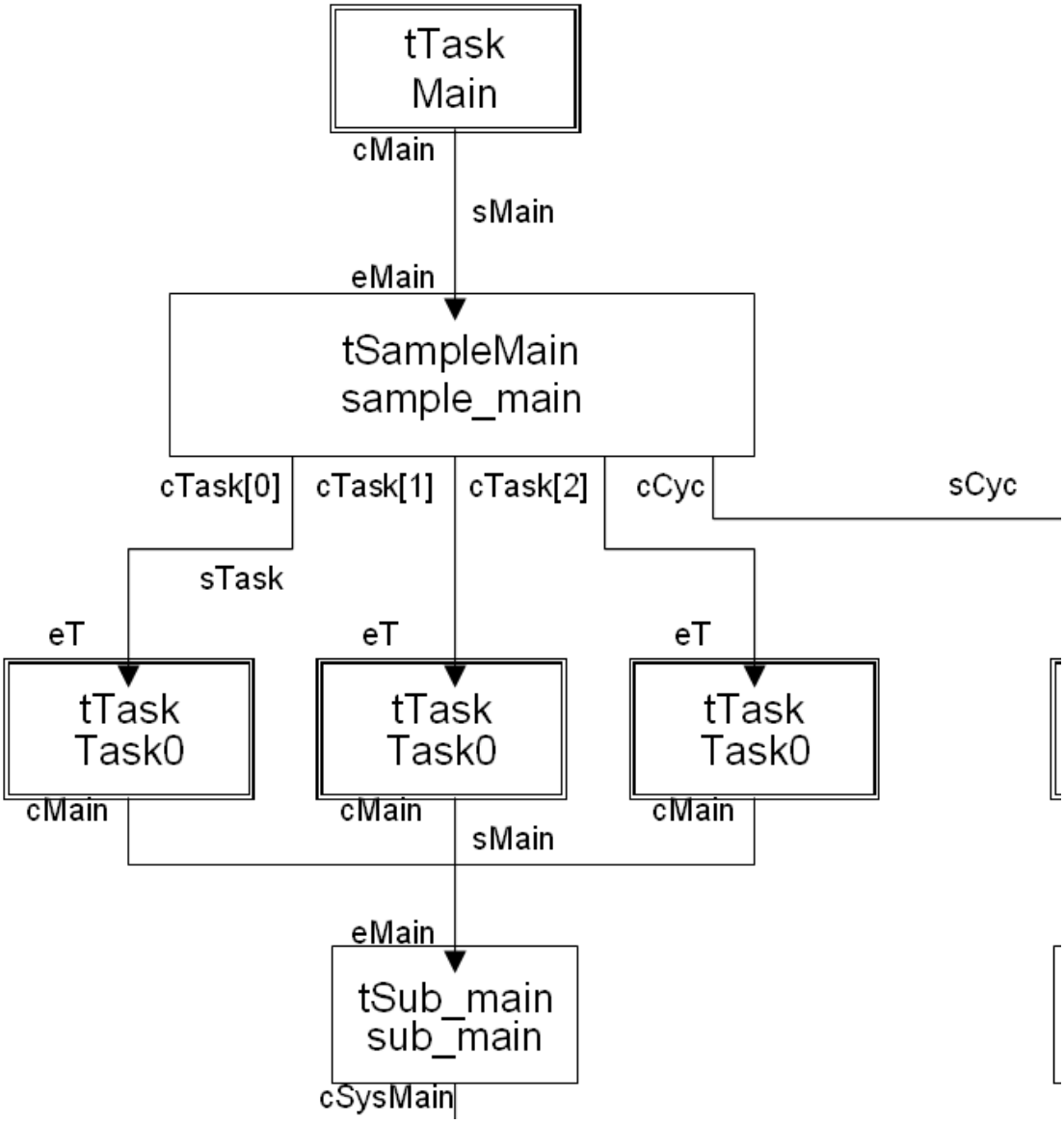
TECS と PlantUML の連携

TECS はコンポーネント組み立て言語の CDL を使って、コンポーネントの関連性が記述でき、ビュアーでコンポーネント図を表示することもできます。
一方、仕様を図示するための規格を定めた UML にもコンポーネント図があり、TECS のコンポーネント図と似たものになっています。UML 図の作成には GUI ツールを使うこともあるが、PlantUML では、テキストファイルで記述することができます。
TECS よりも PlantUML の方が一般に知られているので、PlantUML のコードから CDL を出力するツールを作成することで、TECS の導入が容易になると考えます。また、CDL から PlantUML のコードを出力することで、実装したコードで出力したコンポーネント図を、設計資料に使うことで設計資料が正確で最新の状態を保てると考えます。

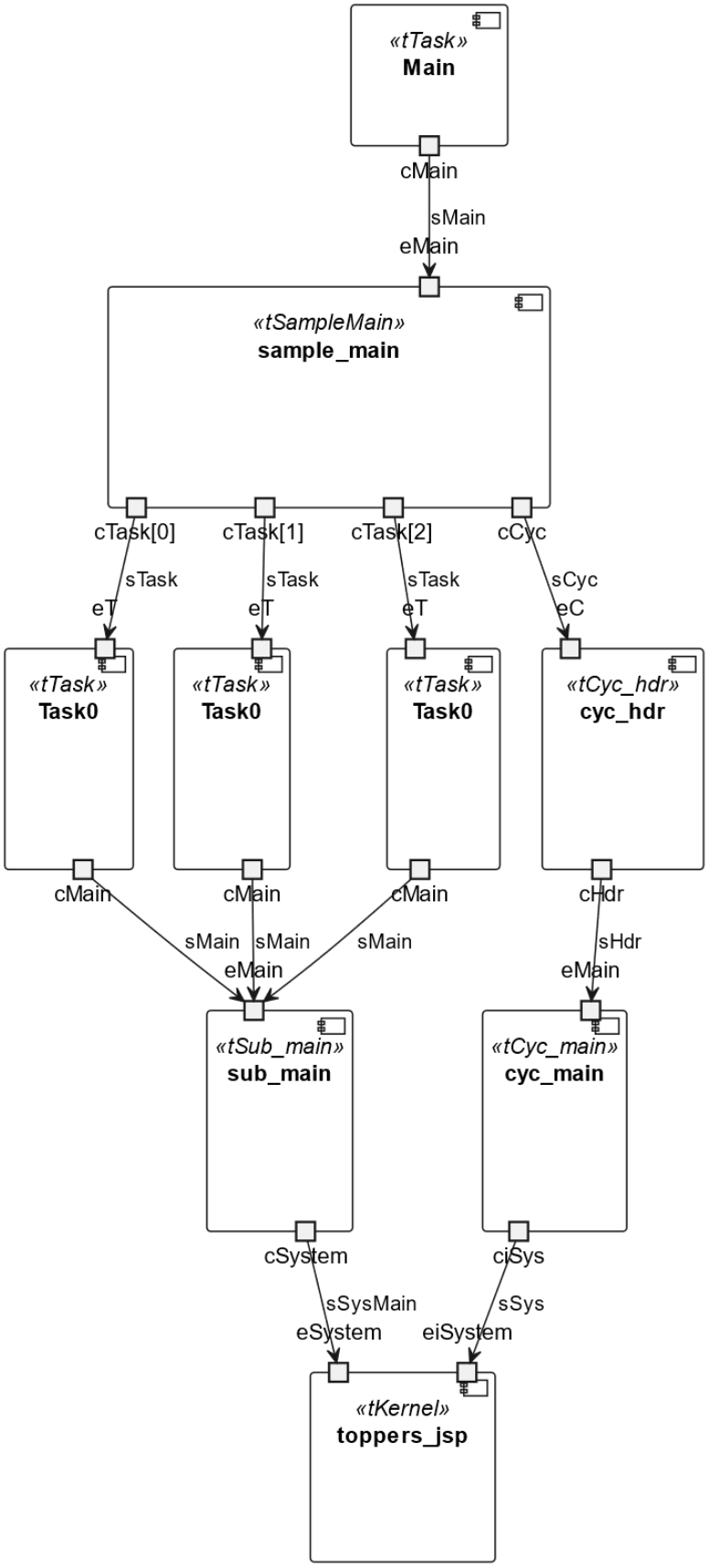
PlantUML：https://plantuml.com/ja/

また、PlantUML の記述は、Visual Studio Code の拡張機能の PlantUML を使用することで、プレビューを見ながらコードを記述することができます。
TECS に応用すれば、プレビューを見ながら CDL を記述できるようになります。
実現すれば、TECS の利用を検討する際の敷居が低くなると考えます。

TECS のコンポーネント図と PlantUML のコンポーネント図を、
TOPPERS/JSP のサンプルプログラムを使って比較します。
参照したのは CodeZine　組込みコンポーネントシステム TECS のコンポーネント図です。



上記のコンポーネント図を、PlantUML の図として書くと次のようになります。


«tTask»
Main
cMain
sMain
eMain
«tSampleMain»
sample_main
cTask[0]
cTask[1]
cTask[2]
cCyc
sTask
sTask
sTask
sCyc
eT
eT
eT
eC
«tTask»
Task0
«tTask»
Task0
«tTask»
Task0
«tCyc_hdr»
cyc_hdr
cMain
cMain
cMain
cHdr
sMain
sMain
sMain
sHdr
eMain
eMain
«tSub_main»
sub_main
«tCyc_main»
cyc_main
cSystem
ciSys
sSysMain
sSys
eSystem
eiSystem
«tKernel»
toppers_jsp

この図の PlantUML のコードは下記の通り。

```
@startuml

component Main <<tTask>> {
    portout cMain
}
component sample_main <<tSampleMain>> {
    portin eMain
    portout "cTask[0]" as cTask0
    portout "cTask[1]" as cTask1
    portout "cTask[2]" as cTask2
    portout cCyc
}
component Task0 as Task1 <<tTask>> {
    portin eT as eTask1
    portout cMain as cMain1
}
component Task0 as Task2 <<tTask>> {
    portin eT as eTask2
    portout cMain as cMain2
}
component Task0 as Task3 <<tTask>> {
    portin eT as eTask3
    portout cMain as cMain3
}
component cyc_hdr <<tCyc_hdr>> {
    portin eC
    portout cHdr as cyc_hdr.cHdr
}
component cyc_main <<tCyc_main>> {
    portin eMain as cyc_main.eMain
    portout ciSys
}
component sub_main <<tSub_main>> {
    portin eMain as sub_main.eMain
```

```
    portout cSystem as sub_main.cSystem
}
component toppers_jsp <<tKernel>> {
    portin eSystem
    portin eiSystem
}

cMain-->eMain : sMain

cTask0-->eTask1 : sTask
cTask1-->eTask2 : sTask
cTask2-->eTask3 : sTask
cCyc-->eC : sCyc

cMain1-->sub_main.eMain : sMain
cMain2-->sub_main.eMain : sMain
cMain3-->sub_main.eMain : sMain

sub_main.cSystem-->eSystem : sSysMain

cyc_hdr.cHdr-->cyc_main.eMain : sHdr
ciSys-->eiSystem : sSys

@enduml
```

CDL から変換するには、PlantUML の記述規則に従うために工夫が必要になるが、同じようなコンポーネント図を書くことが出来ます。

PlantUML のコードパーサーを作ることで、コンポーネントの組み立て方法を読み取り、CDL のテンプレートは出力することが出来きます。シグニチャの定義は PlantUML では記述できないため、CDL を完全に出力することは難しいので、テンプレートとして使います。

既に普及してるツールと連携することで、普及前のツールでも導入を促せると考えます。

以上