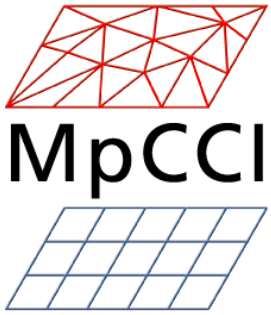
Deliverable 2.6.1:



Erweiterungskonzept
für die ‚Gridifizierung' gekoppelter
MpCCI-Anwendungen

Fraunhofer Institut
Algorithmen und Wissen-
schaftliches Rechnen

Inhaltsverzeichnis

1. Einleitung

Das vorliegende Dokument stellt ein Konzept zur Integration der Kopplungs-Software MpCCI in eine Grid-Infrastruktur da. Hierbei wurde beruecksichtigt, Grid-Komponenten zu waehlen, die mit der D-Grid-Infrastruktur kompatible sind.

Das Dokument gliedert sich in drei Hauptkapitel. Im Kapitel 2 wird der Leser überblicksartig an die Notwendigkeit von Multidisziplinären Simulationen herangeführt. Davon abgeleitet wird in Kapitel 3 der derzeitige Entwicklungsstand von MpCCI dargestellt. Hierbei wird vor allem der MpCCI Code-Adapter detailliert beschrieben. Dieser ist zentraler Bestandteil des Konzeptes zur Integration von MpCCI in ein Grid-Umfeld. Dieses eigentliche Integrationskonzept wird in Kapitel 4 beleuchtet.

2. Multidisziplinäre Simulationen durch Code Kopplung

Simulationen gekoppelter Systeme werden sowohl in der industriellen als auch in der universitären Forschung und Produktentwicklung zunehmend wichtig. Es hat sich gezeigt, dass die Simulation isolierter Problem an Grenzen stoßen, die sich nur durch Kombination verschiedener Simulationen überwinden lassen. Eine Strömung, z. B. eine Luftströmung, erzeugt einen Druckgradienten an einem Körper, beispielsweise einem Flugzeugflügel. Diese Kraft bewirkt eine Deformation des Festkörpers. Auf der anderen Seite verändert diese Deformation die Strömungsbedingungen und somit die Strömung. Neben der beschriebenen Fluid-Structure-Interaction werden Multi-Physics-Simulationen – Simulationen die verschiedene Physik-Subsystem verwenden – beispielsweise in den Bereichen Magneto-Hydrodynamik, Thermische Kopplung oder Simulation von Plasmaentladungen verstärkt gefordert. In all diesen Fällen beeinflusst das eine physikalische Subsystem das Ergebnis eines anderen durch:

- das grundsätzliche Materialverhalten, beispielsweise durch Interpretation eines erhaltenen Druckes als Temperatur,
- Dehnungsabhängigkeit von anderen Ergebnisgrößen, zum Beispiel thermische Dehnung oder Piezo-elektrische Effekte,
- Oberflächen Zug-/Druckkräfte, wie sie von einer Flüssigkeit auf eine Struktur ausgeübt werden,
- Volumenkräfte, wie beispielsweise Wärme erzeugt durch einen Stromverlauf in einer gekoppelten thermo-elektrischen Simulation, und
- geometrische Veränderungen, zum Beispiel für eine Flüssigkeit um eine von ihr deformierten Struktur.

In den meisten Fällen bietet kein einzelnes, proprietäres Simulationsprogramm alle nötigen Features, um ein so gestelltes Multi-Physics-Problem in der gewünschten Weise zu lösen. Daher ist es eine sinnvoll verschiedene Softwarepakete zusammenzubringen, die für spezielle Probleme geschrieben wurden. Mit diesem Ansatz erhält man ein Simulationssystem, das flexibler auf die Anforderungen des Anwenders angepasst werden kann und eine Simulation in der geforderten Qualität ermöglicht.

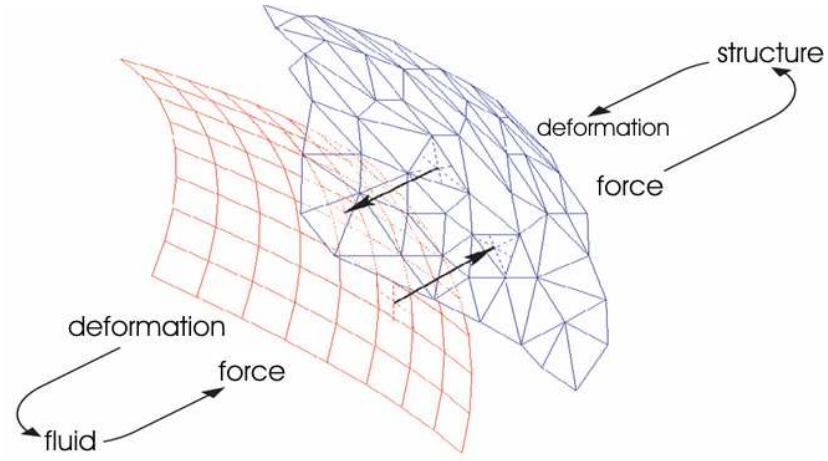


Abbildung 1: Schematisch Darstellung einer Fluid-Structur-Interaction

Als Tool, um oben beschrieben Multi-Physics-Problem zu lösen, wurde MpCCI (Mesh based parallel Code Coupling Interface) am Fraunhofer Institut SCAI als ein Applikations-

unabhängiges Interface zur Kopplung verschiedener Simulationscodes entwickelt. MpCCI bildet eine Software-Umgebung die es ermöglicht Gitterbasierte Daten zwischen zwei oder mehr Simulationscodes innerhalb einer Kopplungsregion auszutauschen. Die notwendigen Interpolationen, aufgrund möglicherweise unterschiedlicher Gitterstrukturen innerhalb der einzelnen Codes, werden ebenfalls durch MpCCI realisiert.
MpCCI erlaubt den Austausch nahezu jeder Art von Daten zwischen den gekoppelten Programmen; z. B. Energie und Impuls, Material Eigenschaften, Gitterdefinitionen oder globalen Größen. Die intrinsischen Details des Datenaustausches werden für den Nutzer transparent abgewickelt. Sind beispielsweise die Simulationscodes als parallele Applikationen verfügbar, wird beim Datenaustausch die Gebietszerlegung von MpCCI berücksichtigt.

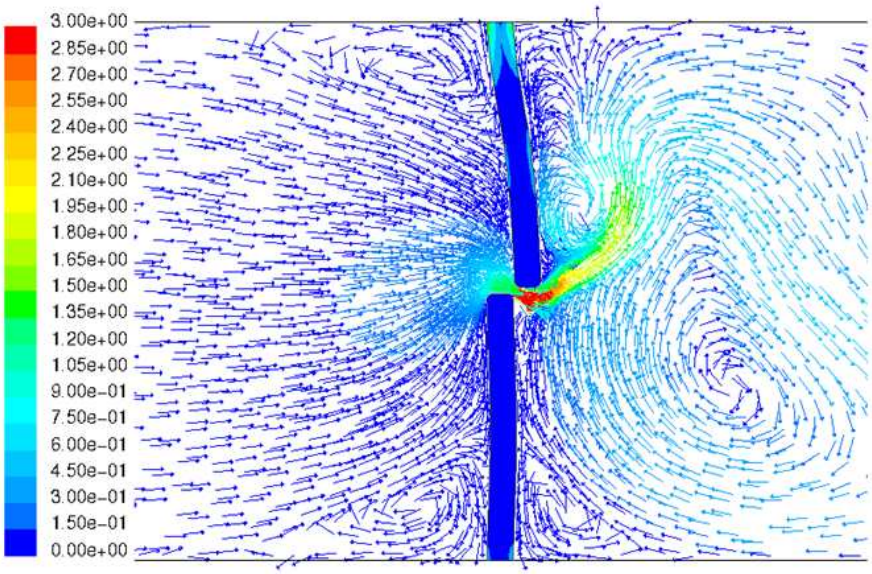


Abbildung 2: Beispiel einer Fluid-Structure-Interaction (Klappe in einer Strömung)

3. MpCCI-Architektur

MpCCI 1.x und 2.0 – Kopplungsbibliothek

Die Entwicklung von MpCCI und seinen Vorgängern begann im Jahr 1996 als Bibliothek, die Informationsaustausch zwischen unterschiedlichen parallelen Simulationscodes ermöglicht. In dem europäischen Project CISPAR und dem deutschen Forschungsprojekt Grissli entstanden die ersten Versionen der Bibliotheken CoCoLib (Coupling Communication Library) und Grissli, die später in der Kopplungsbibliothek MpCCI 1.0 zusammengeführt wurden. Diese erste MpCCI Version wurde im Frühjahr 2001 veröffentlicht und basierte auf MPI (Message Passing Interface) als Kommunikationsschicht. Daher war auch das Design von MpCCI 1.0 ähnlich zu MPI was den Interface-Aufrufe oder das Parallelisierungs- und Kommunikationskonzept betrafen.

Die Hauptidee der Bibliothek MpCCI war die Integration der MpCCI-Aufrufe in direkt den Sourcecode der Simulationsprogramme. Für kommerzielle Codes musste dies zwangsläufig durch den Eigentümer der Programme selbst geschehen. MpCCI wurde beispielsweise erfolgreich in die Codes Ansys, CFX, MSC.Marc, FINE™/Hexa CFD, Permas, StarCD und verschiedene Forschungscodes wie FLOWer oder TAU des DLR integriert. Neben MPI bedingten Kompatibilitätsproblemen, die weitgehend mit der Version MpCCI 2.0 gelöst wurden, brachte diese Vorgehensweise jedoch Nachteile mit sich. Da MpCCI in den Sourcecode der verschiedenen kommerziellen Programme integriert war, mussten alle Modifikationen an MpCCI mit allen Herstellern koordiniert werden. Alle beteiligten Firmen mussten Änderungen an MpCCI zustimmen und diese in ihren Code einbauen. Zugleich musste MpCCI alle Qualitatssicherungs-Schichten der Softwarehersteller durchlaufen. Hierdurch ergab sich ein beträchtlicher Zeitaufwand zur Implementation neuer oder Verbesserung bereits bestehender Features.

MpCCI 3.0 – Multidisziplinäre Simulationsplattform

Aus den oben genannten Gründen wurde 2003 von Fraunhofer SCAI beschlossen das MpCCI Konzept grundlegend zu verändern.
Die meisten kommerziellen Simulationsprogramme erlauben dem Nutzer eigene Features, physikalische Modelle oder Randwerte, mit Hilfe eines offenen API’s (Application Programming Interface), hinzuzufügen. Innerhalb dieser Userroutinen wird der Zugriff auf interne Datenstrukturen durch Subroutinen, globale Variablen oder spezielle Methoden zum Lesen und Schreiben von Daten ermöglicht. MpCCI nutzt diese Fähigkeit der Code-Erweiterungen. Eine User-Subroutine, die in jedem Zeitschritt aufgerufen wird, dient als Verbindung zu MpCCI. Der Vorteil dieser Herangehensweise ist, dass MpCCI nicht länger Programmbestandteil der Simulationssoftware ist, sondern wie jede andere „user defined function“ der Simulation hinzugefügt wird. Daher entfällt die Notwendigkeit einer speziell MpCCI-enable’ten Version der kommerziellen Software.

MpCCI 3.0 – Architektur

Die MpCCI 3.0–Architektur besteht im Wesentlichen aus drei Komponenten:

- dem MpCCI User Interface: Es ermöglicht dem Benutzer auf einfache Weise eine gekoppelte Simulation einzurichten und zu starten.

- dem MpCCI Coupling Server: Das eigentliche „Herz“ von MpCCI. Der Coupling Server beinhaltet die Kommunikation zwischen den Simulationscodes über die MpCCI Code-Adapter, die Nachbarschaftssuche und die Interpolation zwischen den verwendeten Rechengittern.
- den MpCCI Code Adaptern: Schnittstellen zu den (kommerziellen) Codes auf Basis der von den Programm-Herstellern bereitgestellten offenen APIs.

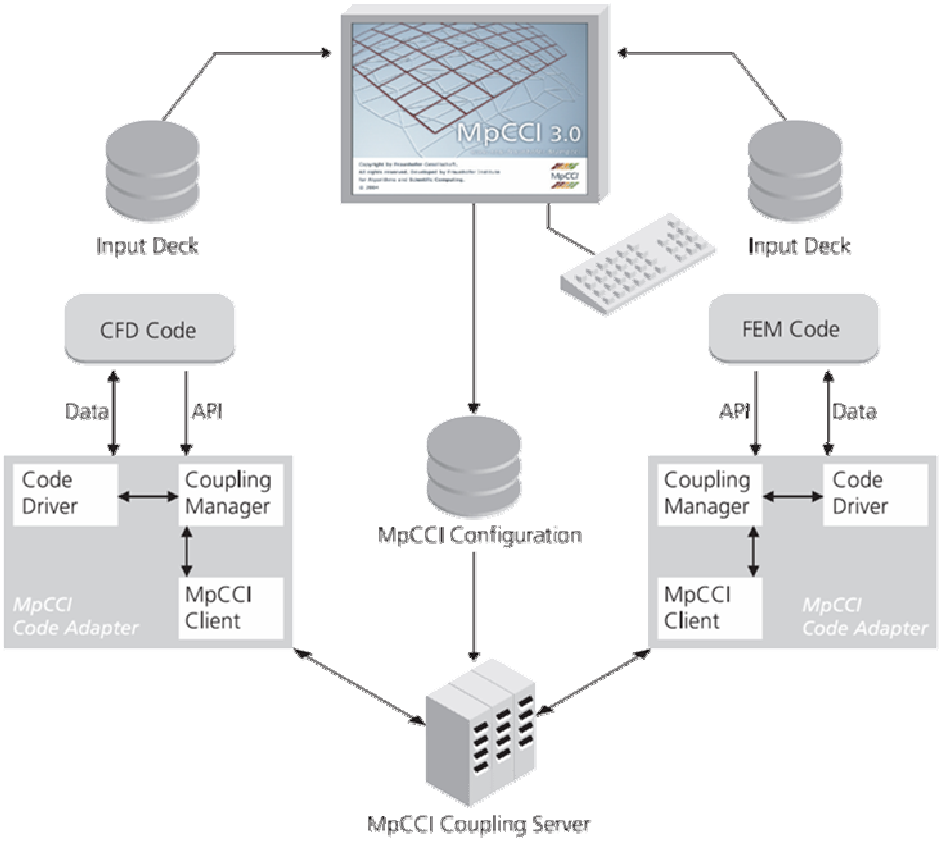


Abbildung 3:Schematische Darstellung der MpCCI – Architektur

MpCCI User Interface

MpCCI wurde als neutrales Interface zwischen verschiedenen (kommerziellen) Simulationscodes entwickelt. Da die Benutzeroberflächen dieser Programme stark unterschiedlich sind, bietet MpCCI selbst eine einheitliche und Code-unabhängige Benutzeroberfläche, die es dem User ermöglicht, alle Kopplungsrelevante Parameter einzustellen. Die grafische Benutzeroberfläche führt den User durch die notwendigen Schritte um eine gekoppelte Simulation aufzusetzen und sie zu starten.

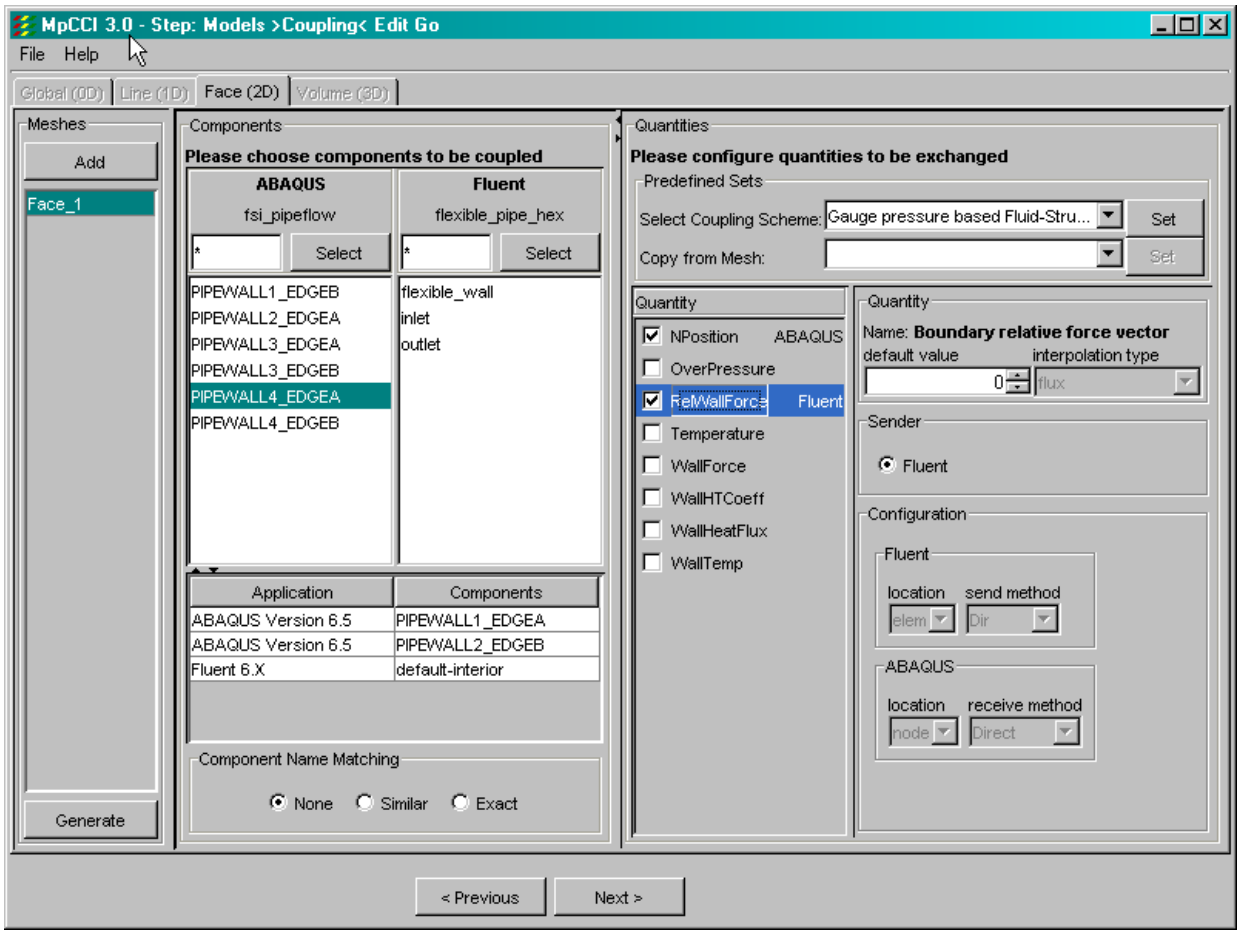


Abbildung 4:MpCCI GUI

MpCCI Coupling Server

Die Aufgabe des Coupling Servers umfasst die drei Gebiete Kommunikation, Nachbarschaftssuche und Interpolation.
Die erste Aufgabe des MpCCI Servers ist es zu bestimmen auf welche Art und Weise die Kopplungsregionen der Simulationsmodelle zusammenpassen. Basierend auf den Gitterinformationen die von den Simulations-Codes und ihren Code-Adaptern zur Verfügung gestellt werden berechnet der Coupling-Server die Nachbarschaftsbeziehungen. Der MpCCI Server wartet nun auf Requests der Simulationscodes. Wenn ein MpCCI-Request von einem Code ausgesandt wird, liest der MpCCI Coupling Server die zugehörigen Daten und schickt sie an den Anfragenden. In diesem Schritt wird, mit Hilfe der Nachbarschaftsinformationen, die notwendige Interpolation der Daten auf das Gitter des fragenden Codes ausgeführt. Je nach Kopplungstyp – Oberflächenkopplung in 3D, Volumenkopplung in 3D, Linienkopplung in 2D ... – werden unterschiedliche Interpolationsverfahren angewendet.

MpCCI Code Adapter

Im MpCCI 3.0 System ermöglichen die Code Adapter die direkte Verbindung zwischen dem jeweiligen Simulationscode und dem MpCCI Coupling Server. Sie nutzen hierzu wie oben beschrieben die von den Code-Herstellern zur Verfügung gestellten API’s und werden statisch oder dynamisch an den Code gelinkt.
Jeder Code-Adapter besteht aus zwei Modulen: dem Coupling Manager und dem Code Driver. Hinzu kommen für jedes Programm noch drei Code-spezifische Skripte um das

Input-File zu scannen, das Simulations-Programm zu starten und zu stoppen, sowie eine Konfigurationsdatei.

Code-Konfigurationsdatei

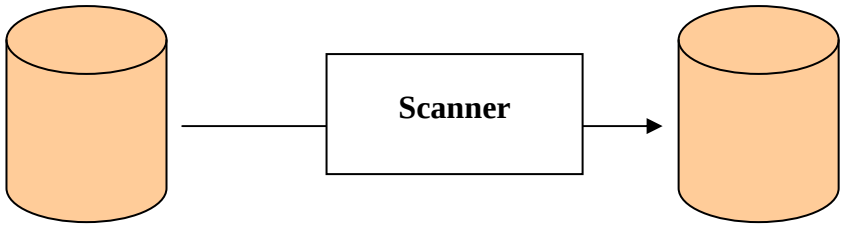
In dieser XML-Datei werden alle für die Integration des Simulations-Programms in MpCCI relevanten Daten und Parameter beschrieben. Hierzu gehören neben direkten Programm-Parametern, wie die zur Verfügung stehenden Programmversionen, auch Angaben für die Beschreibung im MpCCI GUI.

```
<CODE>

  <CodeInfo>
    <Id      type="string"  default="abaqus" />
    <Version type="string"  default="6.5/6.6" />
    <Icon    type="string"  default="abaqus_logo.gif" description="(c) HKS, Inc." />
    <Units   type="string"  default="variable" />
    <Type    type="string"  default="FEM SolidStructure SolidAcoustics SolidThermal" />
  </CodeInfo>


  <ModelsMenuEntries>

    <Release  type="enum" default="latest" description="Select ABAQUS release" >
      <enum value="latest" />
      <enum value="6.6-1" />
      <enum value="6.6-PR8" />
      <enum value="6.6-PR6" />
      <enum value="6.5-1" />
    </Release>

    <ScanMethod  type="enum" default="Scan for all regions" description="Select scan
method" >
      <enum value="Scan for all regions" />
      <enum value="Scan *CO-SIMULATION option only" />
    </ScanMethod>

    <ModelFile type="filename" required="true" default="" description="Select ABAQUS input
deck" >
      <filename suffix=".inp" />
      <filename suffix=".inp.gz" />
      <filename suffix=".INP" />
      <filename suffix=".INP.gz" />
    </ModelFile>

    <Units type="enum" default="SI" description="Select unit system" >
      <enum value="British" />
      <enum value="cgs" />
      <enum value="SI" />
      <enum value="variable" />
    </Units>
```

Ausschnitt eines ABAQUS Konfigurationsfiles

Scanner Skript

Um aus dem Input-File des Simulations-Programms die für die Kopplung zur Verfügung stehenden Größen zu extrahieren ist eine Scanner-Routine notwendig. Dieses code-spezifische Scanner-Skript wird während des Setup der gekoppelten Simulation durch das MpCCI GUI ausgeführt. Die dazu notwendigen Informationen entstammen der oben beschriebenen Konfigurationsdatei.



Input-Datei

MpCCI GUI Datei
.cpl

Durch die Ausführung des Scanner-Skriptes wird das Input-File in ein für das MpCCI GUI lesbares .cpl-File transformiert, in dem nun alle für eine Kopplung verwendbaren Größen stehen.

```
# This file was automatically created for the MpCCI GUI.
#!Scan: "C:\Programme\MpCCI with whitespace\gui\scanners\abaqus2cpl.pl"
#!File: "large.inp"
#!Path: "C:/Dokumente und Einstellungen/cd/Eigene Dateien/DATA-COUPLED-SIM/ABAQUS"
#!Date: "Wed Jan 19 14:37:19 2005"
#!User: "cd"
#!Host: "LAPDEH"
#
# The file contains a list of user defined regions.
# Input deck was scanned for all surfaces and elsets.
# File format:
#  surfaceName                surface  dummy-id
#  elsetName                  elset    dummy-id
#  variableName               global   dummy-id
#
ASSEMBLY_BLOCK-1_BLOCK_TOP      elset          0
ASSEMBLY_BLOCK_BOLT1            surface        1
ASSEMBLY_BLOCK_BOLT10           surface        2
ASSEMBLY_BLOCK_BOLT11           surface        3
ASSEMBLY_BLOCK_BOLT12           surface        4
ASSEMBLY_BLOCK_BOLT2            surface        5
ASSEMBLY_BLOCK_BOLT3            surface        6
ASSEMBLY_BLOCK_BOLT4            surface        7
ASSEMBLY_BLOCK_BOLT5            surface        8
ASSEMBLY_BLOCK_BOLT6            surface        9
ASSEMBLY_BLOCK_BOLT7            surface        10
ASSEMBLY_BLOCK_BOLT8            surface        11
ASSEMBLY_BLOCK_BOLT9            surface        12
ASSEMBLY_BLOCK_TOP              surface        13
ASSEMBLY_BOLT1-1_BOLTS          elset          14
```

Ausschnitt einer ABAQUS - .cpl Datei

Starter Skript

Ist die Konfiguration der Multi-Physics-Simulation beendet, müssen die zugehörigen Simulations-Programme gestartet werden. Hierzu werden Wrapper Skripte verwendet, die notwendige Umgebungsvariablen setzen, den Start der Applikation vorbereiten – z.B. Startup- oder Initialisierungsfiles schreiben – und schließlich das eigentliche Simulations-Programm starten. Auch in diesem Skript werden die im Konfigurationsfile festgelegten Variablen verwendet.

```
<Fluent>
.....
  <Starter>
     <Executable type="string" default="fluent_starter.pl" />
     <Environment>
       <MPCCI_INITIAL_EXCHANGE type="string"
default="%(Fluent.GoMenuEntries.InitialExchange)" />
       <MPCCI_APP_MODEL          type="string"
default="%(Fluent.ModelsMenuEntries.ModelFile)" />
       <MPCCI_CODE_IDSTRING      type="string" default="%(Fluent.CodeInfo.Id)" />
       <MPCCI_FLUENT_VERSION     type="string"
default="%(Fluent.ModelsMenuEntries.Version)" />
       <MPCCI_FLUENT_RELEASE     type="string"
default="%(Fluent.ModelsMenuEntries.Release)" />
       <MPCCI_FLUENT_PROJECT     type="string"
default="%(Fluent.GoMenuEntries.Project)" />
       <MPCCI_FLUENT_GUIOPT      type="string"
default="%(Fluent.GoMenuEntries.GuiOption)" />
       <MPCCI_FLUENT_NPROCS      type="string"
default="%(Fluent.GoMenuEntries.NoProcs)" />
       <MPCCI_FLUENT_PARCOMM     type="string"
default="%(Fluent.GoMenuEntries.ParallelComm)" />
       <MPCCI_FLUENT_JOURNAL     type="string"
default="%(Fluent.GoMenuEntries.JouFile)" />
       <MPCCI_FLUENT_DATFILE     type="string"
default="%(Fluent.GoMenuEntries.DatFile)" />
```

Umgebungsvariablen in einem Fluent-Konfigurationsfile

Stop Skript

Ebenso wie der Start der Applikation wird auch das Beenden der Applikation über ein Wrapper-Skript realisiert, in dem falls notwendig noch ein zusätzliches Cleanup erfolgen kann.

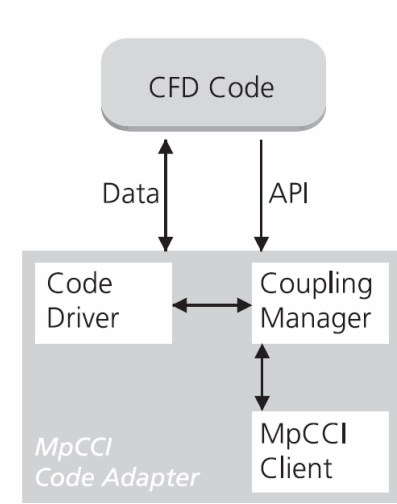


Abbildung 5: MpCCI Code-Adapter-Module

Coupling Manager

Nach dem Start des Simulationscodes wird der Coupling Manager innerhalb des Code Adapters initialisiert. Er erhält vom MpCCI GUI Informationen über die Kopplungsregion und hält Buch über den Status des Simulationscodes. Der Coupling Manager kommuniziert im Weiteren einerseits mit dem MpCCI Coupling Server um Datenaustausch mit anderen Simulations-Codes zu ermöglichen und andererseits mit dem Code Driver um auf die tatsächlichen Simulationsdaten zuzugreifen.

Je nach Zustand der Simulation kann der Coupling Manager

- Gitterinformationen für den MpCCI Coupling Server bereithalten
- Simulationsdaten senden und empfangen.

- Initialisierung

```
CCM_Init (const char *appName, int viaSync,
    CCM_DRIVER_t *appDriver);
```

- Datenaustausch

```
CCM_Transfer(int viaSync, int actBits, int doWait);
```

- Benachrichtigung über Remeshing des Rechengitters

```
CCM_Remesh  (int fullremesh);
```

- Konvergenzcheck an den Partner

```
CCM_Check_convergence ( );
```

Coupling Manager Funktionsaufrufe

Code Driver

Um auf die eigentlichen Simulationsdaten zuzugreifen wird der Code Driver verwendet. Er setzt die generischen Datentransferkommandos des Coupling Manager in die Code-spezifischen Aufrufe der Hersteller-API um.

```
typedef struct _ccm_application_methods_pointer_list {

void (*appendComponents)(int ndefined);
int  (*updateComponents)(void);
void (*selectComponent)(const CC_DESC_t *ccp);
void (*defineGrid)(CC_DESC_t *ccp);
void (*defineCenters)(CC_DESC_t *ccp);
void (*moveNodes)(CC_DESC_t *ccp);
void (*defineClose)(void);

/* methods used to get quantities from the application */
void (*getGlobalValues  )(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getLineNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getLineElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getFaceNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getFaceElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getVoluNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getVoluElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);

/* methods used to store quantities into the application */
void (*putGlobalValues  )(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putLineNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putLineElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putFaceNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putFaceElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putVoluNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putVoluElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
} CCM_DRIVER_t;
```

Code Driver Funktionsaufrufe

4. Konzept zur Grid-Anpassung von MpCCI

Nutzen und Ziele

In der bisherige Arbeitsweise muss ein Benutzer der mit MpCCI ein gekoppelte Multi-Physics-Simulation erstellen will genau Kenntnisse über die ihm zur Verfügung stehenden Ressourcen und die auf ihnen verwendbaren Softwarepakten haben. Es ist ihm nur sehr schwierig möglich auf entfernte Ressourcen zuzugreifen, insbesondere wenn diese nicht in seinem organisatorischen Verantwortungsbereich liegen. So sind User-Accounts zu beantragen, Zugangsberechtigungen zu klären und von Firewalls nicht blockierte Kommunikationskanäle zu ermitteln. Bei der Überwindung dieser Hindernisse kann Grid-Technologie eine Hilfestellung bieten. Hierbei spielen Virtualisierung und Service-Orientierung eine zentrale Rolle. In einer Service-orientierten Sichtweise werden Software-Ressourcen als Services verstanden, die dem Nutzer über klar definierte und von ihm einsehbare Schnittstellen einen Dienst liefern. Für den Benutzer ist es transparent welche Hardware-Ressourcen zur Bereitstellung dieses Dienstes genutzt werden, da er mit dem Dienst nur über die offen gelegten Hardware-unabhängigen Schnittstellen kommuniziert. In weiterführenden Grid-Szenarien kann dem User ein Dienst mit demselben formalen Inhalt unter Einhaltung unterschiedlicher Qualitätskriterien, wie z.B. vorgeschriebene Reaktionszeit, Datendurchsatz, Sicherheitslevel, angeboten werden. Die Entscheidung welche Ausprägung des Dienstes in solch einem Fall in Anspruch genommen wird, kann dann, beispielsweise aufgrund von vom User getroffenen Auswahlkriterien, automatisch erfolgen. Hierzu sind leistungsfähige Ressource Broker erforderlich wie sie momentan in verschiedenen Projekten implementiert werden.
Grid-Technologie bietet darüber hinaus den Vorteil eines komplettes Sicherheits-Environment, das für den Datentransfer und den Zugang zu Ressourcen zuständig ist und eine Unterstützung der Nutzer-Verwaltung auf Ebene von virtuellen Organisationen.
In einem ausgebauten Grid-Umfeld lässt sich schließlich eine Multi-Physics-Simulation in einen größeren Workflow einbinden und auch beispielsweise die Datenhaltung transparent verwalten.

Um diese Vorteile nutzen zu können ist es notwendig MpCCI an Basis-Grid-Technologie anzupassen. In einer ersten Phase soll es MpCCI ermöglicht werden, Service-Ressourcen zu nutzen. In einer zweiten Phase wird MpCCI selbst als Service in einer Grid-infrastruktur nutzbar gemacht. Die genaueren Umsetzungspläne werden im Folgenden erläutert.

Als Basis-Grid-Technologie wird das „Globus Toolkit 4“ Verwendung finden. GTK 4 ist derzeit führender Vertreter im Bereich der Service-orientierten Grid-Middleware-Plattformen und bildet die Basis-Referenz des neu geschaffenen „Web-Service-Resource-Framework“ – Standards (WSRF) fuer Grid-Services. Durch diese Wahl wird eine größtmögliche Akzeptanz und Kompatibilität der zu schaffenden Grid-Anbindung von MpCCI sichergestellt.

Phase 1: Nutzung von Service Ressourcen

In der ersten Phase der Grid-Anbindung von MpCCI müssen Zugriffsmechnismen für Service Ressourcen bereitgestellt werden. Hierzu wird der Code-Adapter und das Skript-Umfeld umgestellt werden.

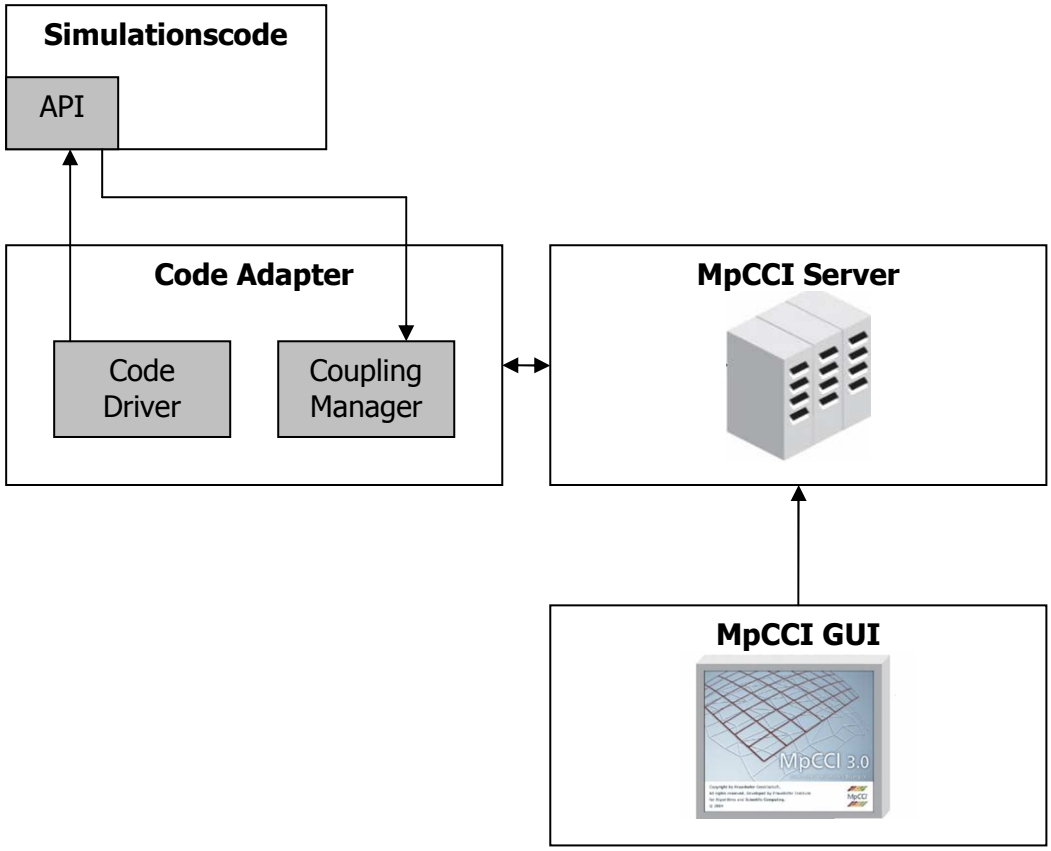


Abbildung 6: Schematische Darstelung der bisherigen MpCCI-Architektur (nur ein Simulationscode)

Da kommerzieller Simulations-Code momentan noch nicht als Service-Resource bereitsteht ist als erster Schritt ein Web-Service-Wrapper, unter Einhaltung des WSRF-Standards, um den eigentlichen Simulations-Code zu implementieren. Dabei werden Methoden zum Starten und Stoppen der Simulation ebenso wie ein Scanner von Simulations-Input-Files in den Web-Service integriert. Diese Methodenaufrufe ersetzen die bisher eingesetzten Skripte und bilden deren oben beschriebenen Schnittstellen zu dem Simulationscode nach. Ebenso wird eine Methode integriert die eine für das MpCCI GUI verständliche XML Beschreibung des Services liefert und das lokal abgelegte Konfigurationsfile ersetzt. Der Aufruf dieser Web-Service Methoden wird in das MpCCI GUI integriert.

Als Hauptbestandteil des Web-Service-Wrapper werden die Methoden des Code-Drivers aus dem Code-Adapter integriert:

```
void (*appendComponents)(int ndefined);
int  (*updateComponents)(void);
void (*selectComponent)(const CC_DESC_t *ccp);
void (*defineGrid)(CC_DESC_t *ccp);
void (*defineCenters)(CC_DESC_t *ccp);
void (*moveNodes)(CC_DESC_t *ccp);
void (*defineClose)(void);

void (*getGlobalValues  )(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getLineNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getLineElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getFaceNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getFaceElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getVoluNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);
void (*getVoluElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, CCIReal *values);

void (*putGlobalValues  )(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putLineNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putLineElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putFaceNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putFaceElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putVoluNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
void (*putVoluElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, const CCIReal
*values);
```

Im verbleibenden Code-Adapter werden diese dann durch Client-Code-Stubs repräsentiert und eine WSRF-Kommunikation zwischen Coupling Manager und Code-Driver-Methoden des Web-Service-Wrapper ermöglicht, die die Funktionalität des ehemaligen Code-Adapter vollständig widerspiegelt.

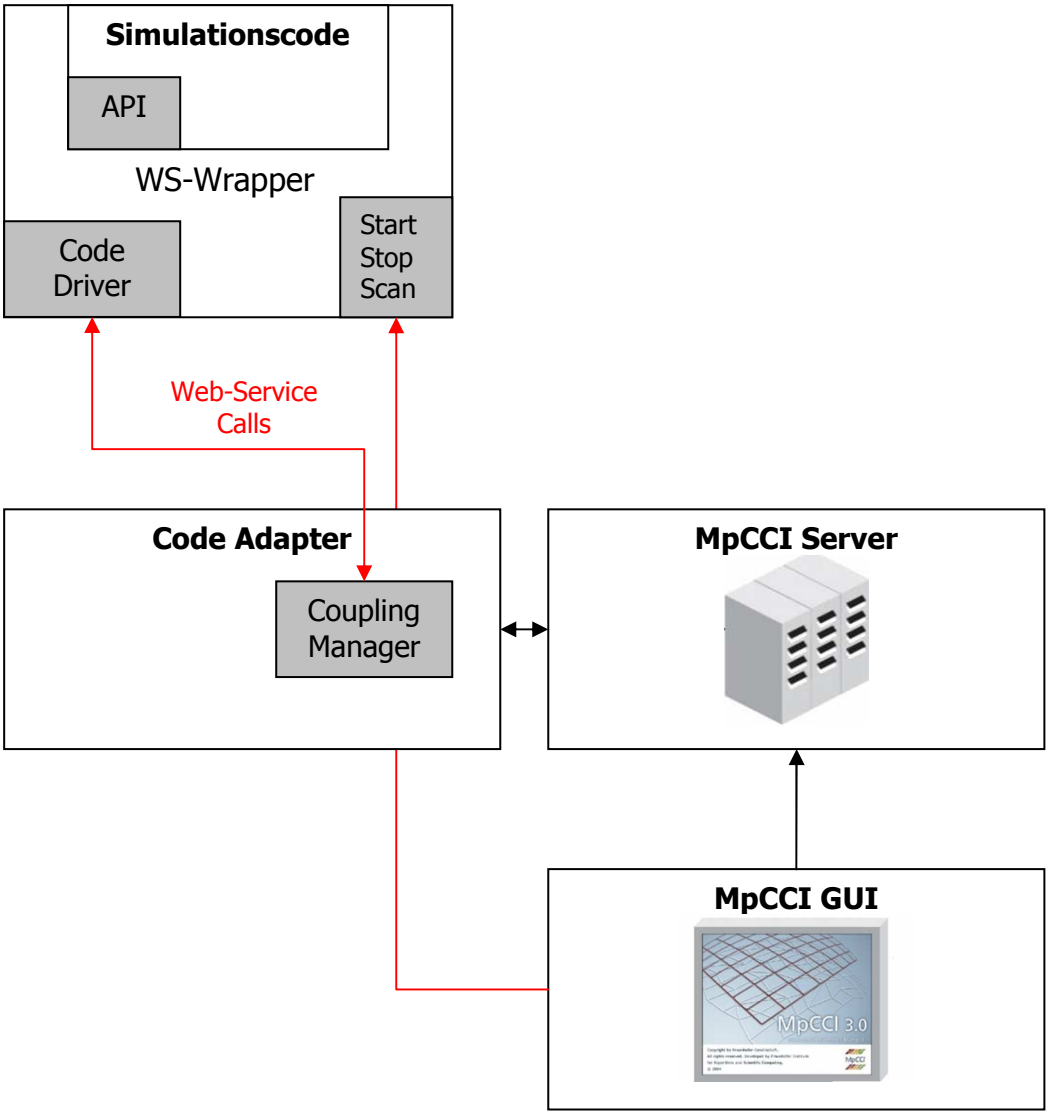


Abbildung 7: Schematische Darstellung der MpCCI Architektur nach Phase 1

Phase 2: MpCCI als Web-Service

Nachdem MpCCI um Funktionalitäten erweitert wurde, um Service Ressourcen nutzen zu können, wird in der zweiten Phase der MpCCI Server selbst als Web-Service realisiert. Dies betrifft im Wesentlichen die Kommunikation mit dem MpCCI GUI. Da der MpCCI Server aus Standpunkt des MpCCI GUIs eine spezielle Applikation – ähnlich eines Simulations-Codes – ist, gestaltet sich die Anpassung analog zum obigen Vorgehen. Um eine reibungslose Integration MpCCI in eine Grid-Umgebung herzustellen ist es jedoch sinnvoll MpCCI ueber ein Grid-Portal zugaenglich zu machen. In diesem werden im Weiteren auch das Datenhandling und die Nutzer-Authentifzierung behandelt. Als sehr funktionelle Loesung hat sich hierbei das GridSphere Portal herausgestellt, das auch vom D-Grid Integrationsprojekt unterstuetzt wird. Daher ist eine Portierung des MpCCI GUIs als GridSphere Plugin in der zweiten Phase vorgesehen.

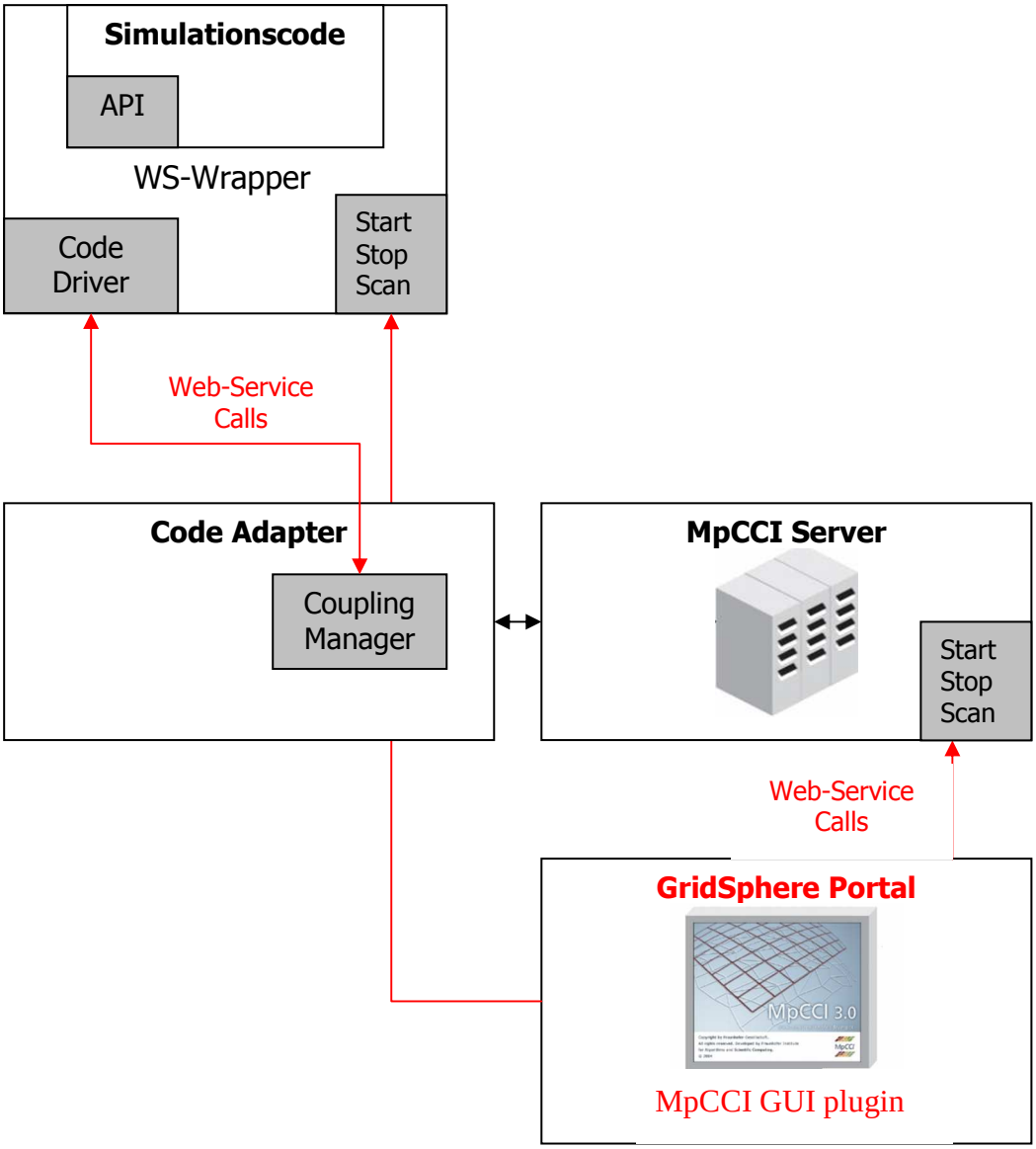


Abbildung 8:Schematische Darstellung der MpCCI Architektur nach Phase 2

5. Literatur

1. Fraunhofer SCAI:
   MpCCI 3.0 Manual – Overview
   MpCCI 3.0 Manual – Technical Reference
   www.scai.fraunhofer.de/mpcci.html , 2005

2. Borja Sotomayor, Lisa Childers:
   Globus Toolkit 4
   Programming Java Servies
   Morgan Kaufmann Publishers, 2006

3. GridSphere-Community:
   Grid Portlets Developer's Guide
   http://www.gridsphere.org/gridsphere/gridsphere?cid=docs , 2006