# Project Partner Contact Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Contact Information</th>
</tr>
</thead>
</table>
| **aicas**        | Fridtjof Siebert  
Haid-und-Neue Strasse 18  
76131 Karlsruhe  
Germany  
Tel: +49 721 66396823  
E-mail: siebert@aicas.com |
| **Bosch**        | Devendra Rai  
Robert Bosch GmbH, Renningen  
70465 Stuttgart  
Germany  
Tel: +49 711 811 15820  
E-mail: devendra.rai@de.bosch.com |
| **CNRS**         | Gilles Sassatelli  
Rue Ada 161  
34392 Montpellier  
France  
Tel: +33 4 674 18690  
E-mail: sassatelli@lirmm.fr |
| **Rheon Media**  | Raj Patel  
20 Leighton Avenue  
Pinner Middlesex HA5 3BW  
United Kingdom  
Tel: +44 7547 162920  
E-mail: raj@rheonmedia.com |
| **The Open Group** | Scott Hansen  
Avenue du Parc de Woluwe 56  
1160 Brussels  
Belgium  
Tel: +32 2 675 1136  
E-mail: s.hansen@opengroup.org |
| **University of Stuttgart** | Bastian Koller  
Nobelstrasse 19  
70569 Stuttgart  
Germany  
Tel: +49 711 68565891  
E-mail: koller@hlrs.de |
| **University of York** | Leandro Indrusiak  
Deramore Lane  
York YO10 5GH  
United Kingdom  
Tel: +44 1904 325 570  
E-mail: leandro.indrusiak@cs.york.ac.uk |
# Table of Contents

1 Introduction .................................................. 2

2 Introduction .................................................. 3
   2.1 Components Common to All ToolFlows .................. 3
   2.2 Components Common to HPC- and Micro-Cloud Use Cases ................................................................ 4
   2.3 Brief Overview of the Components ....................... 4
   2.4 Brief Overview of Integration Results .................. 6

3 DreamCloud Technologies ....................................... 9
   3.1 The Amalthea Model ......................................... 9
      3.1.1 The Hardware Description Section in the Amalthea Model Format ........................................ 10
      3.1.2 The OS Description Section in the Amalthea Model Format .................................................. 10
      3.1.3 The Application Description Section in the Amalthea Model Format .................................... 11
   3.2 Amalthea Parsers ............................................. 12
      3.2.1 The University of York Amalthea Parser .......... 12
      3.2.2 The CNRS Amalthea Parser ......................... 13
   3.3 The Heuristics Module ...................................... 14
   3.4 The CNRS Mapper ........................................... 16
   3.5 Simulators from CNRS ....................................... 17
      3.5.1 The Abstract Simulator ................................. 17
      3.5.2 The Cycle Accurate Simulator ...................... 17
   3.6 Technologies Specific to HPC Cloud Use Case ....... 18
      3.6.1 Block HPC Application ................................. 18
      3.6.2 Block HPC Application Manager ................... 19
      3.6.3 Block HPC Resource Manager ...................... 19
      3.6.4 Monitoring Framework ................................. 19
      3.6.5 Interface to the Heuristics Module ................. 21
   3.7 Technologies Specific to Micro Cloud Use Case ...... 21
      3.7.1 Block Client Controller ................................. 21
      3.7.2 Block Java Dispatcher ................................. 24
      3.7.3 Block Services ........................................... 24
      3.7.4 Block OSGi ................................................ 24
      3.7.5 Block Transcoder ........................................ 24
3.7.6 Block Real Time JVM "Jamaica" ............................................. 25
3.7.7 Interface to the Heuristics Module ....................................... 25

3.8 Technologies Specific to Embedded Cloud Case .......................... 26
  3.8.1 The Model Transformation Block ...................................... 26
  3.8.2 Embedded System Mapper ............................................. 27
  3.8.3 Real Time Embedded System .......................................... 27

4 Conclusions ........................................................................... 29

References .................................................................................. 31
Document Control

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Initial Draft by Robert Bosch GmbH</td>
<td>11.01.2016</td>
</tr>
<tr>
<td>1.0</td>
<td>Final Version</td>
<td>30.04.2016</td>
</tr>
<tr>
<td>1.1</td>
<td>Revised Figures 1 and 5 per Recommendations of the Review Committee</td>
<td>21.10.2016</td>
</tr>
</tbody>
</table>
Executive Summary

This document describes how various methods and tools developed in the framework of DreamCloud are integrated into individual toolchains aimed at Embedded-, HPC-, and Micro-Cloud use cases. The understanding of the interfaces is perhaps the most important step towards an overall understanding of the integrated DreamCloud toolchain. To this end, an abstract description of the interfaces for each component is provided in this document.

This integration report also serves to illustrate the overall value added by the DreamCloud project to the existing challenges in the design and development of complex software systems. For the purpose, brief notes indicating already available integration results from different toolflows are included. Furthermore, references to existing and planned deliverables which cover the evaluation aspects of the toolchain(s) in greater detail are also provided.

In summary, this report confirms that methods and tools developed in the framework of DreamCloud have been seamlessly integrated into toolflows aimed at specific (Embedded-, HPC-, or Micro-Cloud) use cases. Furthermore, available evaluation efforts show that the each toolflow successfully assists the end-user in the design, development, and deployment of complex software systems.

Organization of the Document
Chapter 2 starts with an overview of the toolchain with a brief description of each included component. Thereafter, Chapter 3 describes each component in greater detail, and also points the reader to existing and planned evaluation reports targeted to individual toolflows, followed by a brief conclusion in Chapter 4.
1 Introduction

This report uses a number of terms in the discussion defined as under:

Definition 1. Modes. The application operates under at least two distinct operating modes. The operating mode is determined by the environment (e.g., Start-Stop in a car). Each operating mode specifies a functional behavior of the system, which changes when the mode changes. A change of functional behavior usually results in new tasks being instantiated, some old tasks being dropped, whereas other tasks can continue to run unaffected, or execute with new parameters. In case a change of mode necessitates change of parameters to a running task, the schedulability properties of the given task must be maintained (e.g., no deadline misses).

Definition 2. Runnables, tasks. The smallest software entity to be treated atomically. In case of automotive embedded systems, the corresponding term is a "runnable", whereas HPC systems use the term "task". Several runnables are grouped together to form a "task" in an automotive embedded system, whereas in the case of HPC systems, a task-graph is called as a "job".

Definition 3. Value of a Job. The end-user can provide a notion of value that a system accrues when it executes applications in a specific manner. As an example, executing the application in a way that minimizes its makespan can increase the value of the system. Used primarily in the HPC- and Micro-Cloud use cases.

Definition 4. Amalthea Model. Strictly, Amalthea is a open-source format used for specifying essential characteristics of embedded multi- and many-core software systems. It allows users to specify (i.e., model) their systems using the Amalthea-based specification format, resulting in (colloquially) an Amalthea model.

---

1The term job has also been adopted in the Models of Applications for Dynamic Allocation, see Deliverable D1.3.
2 Introduction

This document describes the individual DreamCloud toolflows used by the industrial partners to evaluate the DreamCloud technologies in their particular use case.

The input to the toolflow is an abstract description of the system (i.e., the Amalthea model), see Figure 1. Note that Figure 1 covers elements necessary for all individual toolflows (i.e., Embedded-, HPC-, and Micro-Cloud use cases).

2.1 Components Common to All ToolFlows

Components: Amalthea Model, Amalthea Parsers, Model Transformation, CNRS Mapper, CNRS Simulators

The input to all individual toolflows is a specification of the system captured in an Amalthea model format. The Amalthea model is then parsed using one of the available parsers. The extracted information is then used by the Heuristics Module to compute resource allocations, and by the CNRS Mapper to configure the simulator. Details on the heuristics used in the Heuristics Module are detailed in Work Package WP2.
In some cases, the specification of the software in the original Amalthea model may be transformed (i.e., made more abstract) using the Model Transformation block. Such a change in the level of abstraction is necessary in case the available design space needs to be explored quickly in order to filter out obviously unfeasible system configurations (e.g., utilization of a processor exceeds 100%), thereby obviating the need to run resource-intensive resource allocation steps.

Detailed performance simulation for a given choice of resource allocations and processor architecture is performed using the Mapper and the simulators developed by CNRS. The simulation framework is generic enough to be used for all toolflows.

2.2 Components Common to HPC- and Micro-Cloud Use Cases

Components: HPC Resource Manager, HPC Application Manager, HPC Application, Real Time Java Virtual Machine, Java Dispatcher, Client Controller, and Transcoder

The request for service is generated by the HPC Application in the case of HPC system, and the Client Controller in the case of Micro-cloud system. Thereafter, the HPC Application Manager communicates with the Heuristics Module in order to provide appropriate resource allocations to the HPC Resource Manager. The HPC Resource Manager also performs runtime monitoring of the system, providing necessary performance data which can be used to improve resource allocations.

For the Micro-Cloud use-case, the Java Dispatcher performs similarly to the HPC Application Manager, whereas the Transcoder performs the service request.

Optionally, the HPC Resource Manager can provide the expected energy consumption by an application, given a resource allocation, and a choice of scheduling discipline used. The energy estimate is then used by the Heuristics Module to refine allocations which may be useful for reducing the overall energy consumption of an application in the HPC- and Micro-Cloud use cases.

2.3 Brief Overview of the Components

A brief overview of the components follows. For details, please refer to Chapter 3.

1. **Amalthea Software Model**: Provided by Bosch, the model provides an abstract description of the system, including details about the hardware platform, operating system, and the description of the application, including tasks, runnables called by each task, associated timing information, and also statistical information about the variations observed in the included timing details.

2. **Model Transformation**: This block transforms the software description in the input Amalthea software model from fine-grained to a coarse-grained software model. That is, in the coarse-grained software model, all execution times summarized at the granularity of tasks, rather than runnables. Such an approach allows the Heuristics Module to compute resource allocations much faster under the constraint that individual tasks must not be split across cores thereby enabling a quick initial assessment of the system.
3. **Amalthea Parsers**: Parsers to extract relevant information from the Amalthea model have been implemented individually by CNRS as well as the University of York on an individual basis. The parser specific to the CNRS is designed to extract system relevant information required by the cycle-accurate simulator, such as the runnables, the tasks, and their dependencies. On the other hand, the University of York use their parser to extract information necessary to compute mapping decisions, such as number of tasks in the system, runnables executed by each task, execution time information for each runnable, as well as dependencies.

4. **Heuristics Module**: The block executes the algorithms developed by the University of York which are responsible computing appropriate resource allocations. For the embedded cloud use case, the allocations are pre-computed for each mode of the system, whereas for Micro- and HPC- Cloud use cases, the allocations are computed online (or pre-computed for online use) and account for most current resource availabilities as well as most current demands for resources.

5. **CNRS Mapper**: This component is used by CNRS to allocate memory and computational resources to runnables as computed by the Heuristics Module. In the HPC context, the mapper is capable of changing allocations for each application as the simulation continues on the simulators developed by CNRS. The changes in resource allocations are driven by the Heuristics Module in response to new demands for resources.

6. **Simulator**: This component is developed by CNRS in order to simulate the application on a platform of choice. CNRS offers choices for abstract functional simulators, as well as cycle-accurate versions. Specific simulators are also available targeted to the choice of interconnect fabric (i.e., crossbar, or network-on-chip) available on the processor of interest.

7. **Micro Cloud Use-Case Components** Refer Figure 1(c).
   
   (a) **Client Controller**: Developed by RheonMedia, the module handles all incoming client requests for video content and provides appropriate responses to deliver the content, if enough resources are available. The client controller is written in C, is wrapped in a Java Native Interface (JNI) wrapper, and built to comply with the Open Services Gateway initiative (OSGi) standard.

   (b) **Java Dispatcher**: The Java dispatcher orchestrates different OSGi compliant components required to appropriately service incoming requests for multimedia. Specifically, the dispatcher receives requests for multimedia streams, allocates computation and communication resources required to service each stream if feasible, and also manages the migration of components between different computational units (e.g., nodes, cores, or processors).

   (c) **Transcoder**: A component responsible for converting streams to different formats according to incoming requests. Different request profiles generate varying demands for resources. There also exists a possibility to use dedicated accelerator resources for speeding up the service to selected requests.

   (d) **Real Time Capable Java Virtual Machine**: Called Jamaica, this component is provided by aicas, and provides support for real-time and high performance computing. It also features a number of extensions which allow tighter control of threads, CPU usage, and memory allocation.
8. **HPC Cloud Use Case Components**: Refer Figure 1(b).

(a) **HPC Application** This component represents a workflow-based HPC application (such as the simulation application MS2 of HLRS) that should be executed on the HPC infrastructure (cluster). The application is specified using either an internal workflow specification or an Amalthea model, as described in the Deliverable D3.1.

(b) **HPC Application Manager** The Application Manager is responsible for steering the execution of the already deployed HPC Applications as well as handling new submission requests (from the end-users). Whenever a task is ready for scheduling (i.e. there are no other “blocking” tasks), the Application Manager requests a new deployment plan from the Heuristic Manager and initiates the deployment of the task on the infrastructure resources.

(c) **HPC Resource Manager**: The Resource Manager keeps track of the status of the infrastructure resources and handles new task allocation, which may come either from the Application Manager, or from the Heuristic Manager.

9. **ES Mapper**: The *Embedded System* mapper allocates memory and processor resources to tasks of the given embedded (specifically, automotive) application, in accordance with the recommendations by the Heuristics Module. Changes to resource allocations are possible due to changes in the dynamics (i.e., *modes*) of the application. Any changes to resource allocations are driven by the Heuristics Module.

10. **Real Time Embedded Systems** Representative examples of automotive engine control applications are provided by Bosch, specified in the Amalthea format. The provided examples consist of hundreds of runnables contained in a dozen or more tasks.

2.4 **Brief Overview of Integration Results**

This section provides a quick overview of the value contributed to various use-cases by the methods and tools developed with the framework of DreamCloud.

**Embedded Cloud Use Case**

For the embedded cloud use case, the Heuristics Module forms the most important component developed within the framework of DreamCloud. Specifically, the Heuristics Module is used to quickly generate *feasible* resource allocations which ensure that all tasks meet their respective timing deadlines, see Figure 2. Without the availability of the Heuristics Module, it is infeasible to compute such resource allocations manually (or by brute-force) for a representative engine control system with thousands of runnables and dozens of tasks, see Deliverable D6.2 for more details.

**Micro- and HPC- Cloud Use Cases**

For both Micro- and HPC-cloud use cases, methods and tools developed within the framework of DreamCloud allow the user to continuously configure the system in order to extract maximum *value*, and/or lower the overall energy consumption. Colloquially, the notion of *value* gained by the system
For the embedded cloud use-case, the Heuristics Module can be used to quickly compute feasible resource allocations.

The DreamCloud toolchain allows the user to attach a notion of value gained by the system. In this figure, the value gained by the system is inversely proportional to the makespan of the given job.

is higher if it can it can complete a job faster, see Figure 3. Technically, the value of gained by the system is higher if the application is executed in a manner which lowers its makespan.

The Heuristics Module provides appropriate allocation of resources based on the choice of metric to optimize for the system under consideration, see Figure 4 for an example. Referring Figure 4, the system is configured with different resource allocations targeted to either increase the overall value obtained from a system, or decrease the total energy consumption, or both, see Deliverable D2.3 for more details.
Figure 4: The DreamCloud toolchain allows the user to allocate resources in a way which maximizes the overall value gained from the system, or decrease the total energy consumption, or both. Key: NBA: Non-Profiling Based Approach, FCPS: Fixing Cores Power States, FTPS: Fixing Tasks Power States, PBA: Profiling Based Approach. See Deliverable D2.3 for details.
3 DreamCloud Technologies

This chapter describes in detail all software components which have been contributed by different Dreamcloud partners. The overall component-level overview is reproduced here again, see Figure 5.

3.1 The Amalthea Model

The Amalthea model conforms to the specifications laid down by the EU-funded AMALTHEA4public project, see [6]. The Amalthea model, and the associated tools are intended to used by "tool vendors,
engineering companies and other suppliers in the toolchain to efficiently integrate their products and expertise.\textsuperscript{2} The Amalthea model format supports the description of the various system components (e.g., hardware, operating system, tasks, functions, execution times), see Figure 6.

3.1.1 The Hardware Description Section in the Amalthea Model Format

A snapshot of a simplified hardware section of the Amalthea model is shown in Figure 7.

\begin{verbatim}
<hwModel xml:id="H08bBN9NNe0ETou3oNnhPQ">
  <coreTypes xml:id="H08bBd9NNe0ETou3oNnhPQ" name="Default" instructionsPerCycle="1"/>
  <system xml:id="H08bBt9NNe0ETou3oNnhPQ" name="Democar">
    <ecus xml:id="H08bB99NNe0ETou3oNnhPQ" name="ECU_1">
      <quartzes xml:id="H08bC99NNe0ETou3oNnhPQ" name="QuartzOscillator_1" frequency="200000000" type="STATIC"/>
      <microcontrollers xml:id="H08bCd9NNe0ETou3oNnhPQ" name="Processor_1">
        <ports xsi:type="hw:ComplexPort" xml:id="H08bC99NNe0ETou3oNnhPQ" name="Mem_1_port" bitWidth="16" writeCycles="10" readCycles="10"/>
      </microcontrollers>
    </ecus>
    <cores xml:id="H08bD9NNe0ETou3oNnhPQ" name="Core_1" coreType="H08bBd9NNe0ETou3oNnhPQ">
      <prescaler xml:id="H08bD9NNe0ETou3oNnhPQ" name="Core_1_prescaler" quartz="H08bC99NNe0ETou3oNnhPQ"/>
    </cores>
  </system>
</hwModel>
\end{verbatim}

\textbf{Figure 7:} Example of the hardware section of the obfuscated Amalthea model

Notice that the hardware description block contains details such as the number of instructions executed by the processor per clock tick, the frequency of the quartz oscillator, the description of the attached memory module(s) together with read- and write-cycle costs, and the number of cores present in the considered processor (e.g., ECU_1).

3.1.2 The OS Description Section in the Amalthea Model Format

A snapshot of a simplified OS section of the Amalthea model is shown in Figure 8.

The Amalthea model allows for the specification of the scheduling algorithm used (e.g., priority preemptive implemented by OSEK), associated scheduling overheads (e.g., due to context switches).

\textsuperscript{2}taken from Amalthea 4 public website: http://www.amalthea-project.org
3.1.3 The Application Description Section in the Amalthea Model Format

The Amalthea model format allows for the specification of tasks, together with the associated periods (if applicable), runnables called by each task, together with the observed execution times of each runnable (e.g., best-case execution times, worst-case execution times), and also statistical functions which describe the variations in the observed execution times, e.g., Weibull distribution. A section of a simplified Amalthea model describing some of the parameters is shown in Figure 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<tasks xml:id="H08Yg9NNeEOTou3oNmhPQ" name="Task_10MS" priority="10" stimuli="H1Fid9NNeEOTou3oNmhPQ" preemption="preemptive" multipleTaskActivationLimit="10">
  <deadline xml:id="H08Yh9NNeEOTou3oNmhPQ" value="10" unit="ms"/>
  <callGraph xml:id="H08Yh9NNeEOTou3oNmhPQ">
    <calls xml:id="sw:TaskRunnableCall" xml:id="H08Yht9NNeEOTou3oNmhPQ" runnable="H08Yh9NNeEOTou3oNmhPQ"/>
    <calls xml:id="sw:TaskRunnableCall" xml:id="H08Yht9NNeEOTou3oNmhPQ" runnable="H08Yh9NNeEOTou3oNmhPQ"/>
  </callGraph>
</tasks>
```

**Figure 9:** Example of the application section of the obfuscated Amalthea model

The activities, and execution times of each runnable can also be specified in details, including statistical variations, see Figure 10.

As shown in Figure 10, the runnable ABSCalculation is observed to have a minimum execution time of 72,000 instructions (5ns per instruction on a processor clocked at 200MHz), a worst-case execution time equivalent to 88,000 instructions, with a mean execution time equivalent to 80,000 instructions, along with statistical information such as pRemainPromille, the probability that the execution time of a runnable exceeds its worst-case time.
3.2 Amalthea Parsers

Both University of York and CNRS have developed custom parsers to read in Amalthea models supplied by Bosch, as detailed below:

3.2.1 The University of York Amalthea Parser

The abstract overview of the Amalthea parser used by the University of York is shown in Figure 11.

![Figure 11: Abstract view of the Amalthea parser developed by the University of York. The dependency provided by Bosch is also shown.](image)

**Input:**
The input comes from the specification of system contained in the Amalthea model. Specifically:

1. The description of the hardware used in the system.
2. The description of the operating system used in the system.
3. The description of the application software used in the system.
Output:

1. Hardware description: Type, total number of cores available in the processor, and the frequency of the oscillator.

2. Timing constraints of the application: e.g., task periods, and time instants when the application mode changes (i.e., application dynamics).

3. Dependency description: in the form of task-graphs.

4. Application description: number of runnables, number of labels, number of stimulus objects, number of tasks, and associated execution times.

This information is used as an input to the Heuristics Module for computing resource allocations, see Deliverable D5.1 for details.

3.2.2 The CNRS Amalthea Parser

The parser developed by CNRS is based on the parser by University of York with a few enhancements as detailed below. Enhancements to the parser are shown in the italic typeface. The abstract view of the Amalthea parser used by CNRS is shown in Figure 12.

**Figure 12:** Abstract view of the Amalthea parser developed by the University of York. The dependency provided by Bosch is also shown.

The parser developed by CNRS is based on the parser by University of York with a few enhancements as detailed below. Enhancements to the parser are shown in the italic typeface. The abstract view of the Amalthea parser used by CNRS is shown in Figure 12.

**Inputs:**

1. The description of the hardware used in the system.

2. The description of the application software used in the system.
Outputs:

1. Hardware description: Type, total number of cores available in the processor, and the frequency of the oscillator.
2. Timing constraints of the application: e.g., task periods, and time instants when the application mode changes (i.e., application dynamics).
3. Dependency description: in the form of task-graphs.
4. Application description: number of runnables, number of labels, number of stimulus objects, number of tasks, and associated execution times, event types (e.g., aperiodic), new runnable attributes (e.g., order of task release times),

This information is then used to configure the simulator.

3.3 The Heuristics Module

Figure 13: Abstract view of the Heuristic Module. Necessary dependencies are also shown.

Developed as part of Work Package 2, the Heuristics Module is responsible computing appropriate resource allocations, i.e., runnable-to-core mappings, mappings of labels-to-memories, as well as suggesting a scheduling discipline, see Figure 13.

The Heuristics Module leverages concepts introduced in the University of York’s work on Interval Algebra for computing resource allocations on multiprocessor systems. These heuristics compute the appropriate number of processors required to run the given application, as well as the scheduling discipline to be used, under the constraint that no task in the system misses its deadline, see [5]. The
Heuristics Module is also capable of estimating the energy cost associated with a given resource allocation. The Heuristics Module exploits its estimates of energy costs to compute energy efficient resource allocations, or even refine allocations at runtime. Published results are already available demonstrating that the resource allocations generated by the Heuristics Module can be used to reduce makespan of the schedule, as well as decrease the energy consumption in the context of HPC systems, see [5]. Details about the heuristics can also be found in deliverables D2.1, D2.2, D2.3, and D3.4.

In the context of embedded real-time systems, the Heuristics Module pre-computes resource allocations for each operating mode of the system. At the instants of mode changes, the pre-computed allocations are simply applied to the runtime system, e.g., the Timing-Architects simulator. Such a pre-computation strategy is usually applied in the context of embedded systems with dynamics since such systems usually do not have sufficient resources to explore the available design space at runtime, see [9].

In contrast, for soft real-time use cases, e.g. for HPC, the Heuristics Module is part of the runtime execution framework. The Heuristics Module executes concurrently with the managed platform, dynamically mapping jobs to cores. In such cases, it leverages up-to-date information about application execution profiles acquired by the available performance monitors, and operates in a closed-loop setup. In order to fine-tune heuristics prior to deployment, the Heuristics Module can be connected to a platform simulator (see next subsection) in order to evaluate the impact of different resource allocations in response to varying demands for resources by different applications.

The interface information for the Heuristics Module is summarized as:

**Inputs:**
The inputs are described on a component-basis, i.e., one line for all inputs coming from a given input component.

1. Description of the system: application, dependencies, timing constraints, and hardware information provided by the Amalthea parser.
2. Mode change timings provided by the Embedded Cloud use case.
3. Monitoring and performance data, request to schedule new jobs, and workflow specification provided by the HPC- and Micro-Cloud use cases.

**Outputs:**

1. Allocation of Processor Resources:
   (a) Embedded-Cloud use case: allocation of cores to runnables.
   (b) HPC-Cloud use case: allocation of jobs to nodes, processors and cores.
   (c) Micro-Cloud use case: allocation of jobs to nodes, processors, and cores.
2. Allocation of Memory Resources:
   (a) Embedded-Cloud use case: allocation of memories to labels.
   (b) HPC- and Micro-cloud use case: allocation of communication flows to interconnects (indirectly through the allocation of jobs).
3.4 The CNRS Mapper

The mapper decides where a particular runnable instance should be allocated on the multicore architecture, see Figure 14. It answers to requests for resources to be allocated to runnables or labels (embedded context) or jobs (HPC context). These requests contain the identifier of the object for which resources need to be allocated, and the mapper responds with the identifier of the particular core or memory of the platform responsible for executing the runnable in question.

The interfaces of the Dynamic Mapping Module are summarized as under:

**Input:**

1. The name and ID of the label which needs to be allocated memory resources.
2. The name and ID of the runnable or job which needs to be allocated computing resources.
3. The execution time of a completed runnable.

**Output:**

1. The memory resource to which the given label must be allocated.
2. The computing resource which will execute the given runnable or job.

---

Figure 14: Abstract view of the Heuristic Module. Necessary dependencies are also shown.
Figure 15: Abstract view of the CNRS Simulator Module. Necessary dependencies are also shown.

3.5 Simulators from CNRS

CNRS provides two types of simulators which are being used to validate the mapping and scheduling options computed by the Heuristics Manager.

3.5.1 The Abstract Simulator

Also called McSim-TLM-NoC, the abstract transactional simulator for multi-core platforms considers transaction-level core models interconnected by a packet-based communication network. During the execution, the cores can support different scheduling techniques, such as Earlier Deadline First (EDF), Rate Monotonic or Application Inherited Task Prioritization. The abstract simulator designated AMALTHEA-SimGrid is also able to interface with the SimGrid Engine for simulating large scale distributed systems, such as those commonly used in High Performance Computing (HPC), see [4].

The abstract simulators are also capable of estimating the energy consumed by an application. The energy consumption of an application is estimated using the McPAT simulator, see [3, 7]. The energy information is derived by analyzing the instruction execution times together with the power estimates generated by the McPAT simulator.

The abstract simulators are fast, but since it is based on transaction level models of computer architecture(s), it is not very accurate, specifically as far as estimating timing and power performance of communication network(s) on the chip.

Further details on the abstract simulator can be found in Work Package 5, deliverable 5.2.

3.5.2 The Cycle Accurate Simulator

The cycle accurate simulator improves the simulation accuracy of abstract simulator, especially in the simulation of network elements, e.g. the Network-On-Chip or the Crossbar. Specifically, all properties of the network infrastructure in the selected processor are modeled at the cycle-accurate granularity.
Both crossbar and network-on-chip type communication infrastructures are supported. Specifically, the cycle-accurate simulator targeted to NoC-type interconnect is designated McSim-CA-NoC, whereas the version targeted to a crossbar-type interconnect is designated McSim-CA-Xbar.

Obviously, the improved accuracy of the Cycle Accurate Simulator over the Abstract Simulator comes with significantly increased demands for computational resources as well as memory.

Further details about the Cycle Accurate Simulator can be found in Work Package 5, Deliverable 5.3.

### 3.6 Technologies Specific to HPC Cloud Use Case

A workflow management system for HPC-Cloud scientific applications was implemented based on the DreamCloud reference architecture leveraging the service-based architecture, as depicted by Figure 16. The system is described in detail in the WP3 deliverable D3.3. Below we provide a short summary of the main interactions between the components of the architecture. Also refer Figure 5 for the overview and integration with the DreamCloud toolchain.

#### 3.6.1 Block HPC Application

**Workflow Manager**

The user’s workflow submission requests are first processed by the Workflow Manager which is designed as an extension to the native HPC resource manager (such as pbs/torque) and is used to manage the resources within a cluster, i.e., a fixed set of compute nodes available to all workflows in the...
HPC application configuration. Workflow Manager is targeted to meet the consolidated requirements of all workflows that have been submitted.

3.6.2 Block **HPC Application Manager**

**Scheduling Interface**
Scheduling Interface of the Workflow Manager handles user requests and passes them further on to the Scheduling Advisor which in turn communicates with Heuristics Module.

**Scheduling Advisor**
Scheduling Advisor is a part of the system designed to generate and handle requests between Workflow Manager, Heuristics Module and Resource Manager. After extracting task-specific information from the original job description, it generates a request for the deployment plan to be processed by the Heuristics Module. Scheduling Advisor also communicates with the Resource Manager to query the availability of computational resources (such as CPUs).

**Progress Tracker**
The Progress Tracker component keeps track of the tasks currently being executed on the target system. Each time a new task is started for execution, or is completed, the Workflow Manager updates the Progress Tracker about the state of the task. Progress Tracker in turn informs the Monitoring Framework about the current state of the execution process.

3.6.3 Block **HPC Resource Manager**

**Resource Manager**
Resource Manager keeps tabs on available resources as well as their current state of configuration. It provides responses to availability queries initiated by the Scheduling Advisor and updates the internal resource availability database as a part of the execution and profiling process.

**Monitoring Framework**
The Monitoring Framework component collects and processes execution related statistics which is then sent to the Heuristics Module. Deliverable D6.5 will provide more details on the implementation of the Workflow Manager components as well as application specification and implementation results. As for the time of this deliverable submission, all HPC-Cloud workflow manager’s components are available for the early adoption by the application (scientific workflow) developers.

3.6.4 Monitoring Framework

In-depth knowledge of application and infrastructure performance and power properties is essential for applying heuristics for dynamic applications scheduling. This aim is fulfilled by means of the monitoring platform, introduced in the deliverable D3.1. The DreamCloud monitoring platform
Listing 1: Code instrumentation via a C library. Firstly, the URL to the web server as well as an experiment ID that associates the sampled data with the current execution has to be passed to the method `atom_init`. Secondly, progress information is sent via calling `atom_update`.

is based on the monitoring solution from the EXCESS project and is able to collect the following categories of metrics:

1. **Computation**: utilization of devices like CPUs and Memory by leveraging PAPI hardware counters (through PAPI-C API)

2. **I/O**: network and disk utilization (through linux tools like Iostat)

3. **Energy**: energy consumption on the per-task basis (in the contents of HPC workflow) by means of interfaces like RAPL and also directly from the dedicated power measurement system (as part of the EXCESS testbed)

4. **User-defined**: any metric that the users might want to collect in their application and save to the monitoring database.

The computation, I/O, and energy metrics are collected automatically on the infrastructure nodes. Collection of the user-defined metrics is enabled by means of a special API, exposed to the programmer (available for Java, C, and Fortran languages). The pre-defined metric `progress` is intended to track the progress of the application (i.e., the percentage of the completed work in the application) and helps improve the resource allocation. The application developers are strongly advised to specify this metric in the applications (as shown by an example in Listing 1). All metrics, including the user-defined ones, are stored in a unified way in a monitoring database (served by the open-source Elasticsearch technology). In order to get access to the stored profiles, the monitoring platform exposes a RESTful API that allows obtaining any metric for the specific task (based on a unique task id and the metrics’s name). Listing 2 shows an example of the query to the monitoring database and a response for the "memory" metric.
curl -XGET http://mf.excess-project.eu:3030/dreamcloud/mf/profiles/sample.workflow/t2.1/AU3TtO0aYHjgymAd2i5T

[...

Listing 2: Request to retrieve a performance profile for a given combination of workflow ID (sample.workflow), task ID (T2.1), and experiment ID (AU3TtO0aYHjgymAd2i5T).

3.6.5 Interface to the Heuristics Module

Heuristics Module

Heuristic Module (insofar as scheduling of jobs is concerned) is a "black-box" component that accepts requests by the Scheduling Advisor for the deployment plans of each individual job (i.e. which node, processor and core it is allocated to). The Heuristics Module produces such plans based on data received previously from the Scheduling Advisor and the Monitoring Framework: workload model, current resource availability, supplied energy-awareness criteria and historical execution statistics collected during the initial learning phase and in the production phase. Once a deployment plan is issued, a job will be executed on the allocated node and processor until it completes. See Section 3.3 for more details.

3.7 Technologies Specific to Micro Cloud Use Case

The components specifically targeted to the Micro-Cloud use case are shown in Figure 17. Detailed control and data flow is shown in the sequence diagram, see Figure 18.

3.7.1 Block Client Controller

Developed by RheonMedia, this component handles all incoming client requests for video content and provides appropriate responses to deliver the content, if enough resources are available. The Client
controller is written in C, but has a JNI wrapper and built as an OSGi Bundle. The Client Controller provides external interfaces to clients and communicates with the Java Dispatcher for servicing new requests, updates and closure of requested streams. The Client Controller executes continuously on a specific node along with an instance of the Java Dispatcher and the Heuristics Manager. Incoming channel requests include information about the client, i.e. a client identifier, the device type and the requested channel. These parameters provide information to the Heuristics Module which in turn determines if the request can be fulfilled.

The JNI wrapper provides APIs that can be executed by the Java Dispatcher to provide responses to clients. A brief summary of available APIs (informally described as actions) is provided in Table 1.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request New Resource</td>
<td>The Client Controller request a computing resource(s) (e.g., a transcoder) for servicing a new request</td>
</tr>
<tr>
<td>Response to Resource Request</td>
<td>The Java dispatcher informs whether a resource can be provided or not.</td>
</tr>
<tr>
<td>Locate the Resource (e.g., transcoder) for an Approved Request</td>
<td>The location (e.g., URI) of the resource in case of an approved request</td>
</tr>
<tr>
<td>Find Status of Resource Request</td>
<td>Query the status of a request for a resource</td>
</tr>
<tr>
<td>Refuse Request Explicitly</td>
<td>Deny servicing a request (even when resources are available)</td>
</tr>
<tr>
<td>Terminate a Completed Request</td>
<td>Bring back the resource servicing a completed request to the pool of available resources</td>
</tr>
</tbody>
</table>

**Table 1:** A brief summary of APIs (informally described as actions) available to the Client Controller.
Figure 18: Overview of the control and dataflow between components developed for the Micro-Cloud use case.
3.7.2 **Block Java Dispatcher**

The Java Dispatcher coordinates different nodes, making a cluster out of them. It provides information about the available resources over all the nodes and passes deployment and migration orders to the right devices. It also provides the Heuristics Module with system information gathered from the real time Java Virtual machine, *Jamaica*. The dispatcher manages incoming requests and provides the intercommunication and management of other bundles.

Specifically, the dispatcher can take the following actions, available as an API are succinctly listed in Table 2.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bind Resource to Request</td>
<td>Allocate a resource (e.g., an instance of the transcoding) and bind it to an incoming request</td>
</tr>
<tr>
<td>Close a Request</td>
<td>Inform the Java Dispatcher that a request has been cancelled, and corresponding resources can be freed up.</td>
</tr>
<tr>
<td>Deploy Transcoder</td>
<td>Deploys an instance of an OSGi Transcoder bundle and starts up with parameters passed by the client controller on a given node.</td>
</tr>
<tr>
<td>Undo Deploy Transcoder</td>
<td>Actual API is <code>DeployTranscoderRefused</code>.</td>
</tr>
<tr>
<td>Get Cluster Information</td>
<td>Get the status of the computing resources (%used, %free, etc.)</td>
</tr>
</tbody>
</table>

**Table 2:** A brief summary of APIs (informally described as actions) available to the Java Dispatcher.

3.7.3 **Block Services**

These are services providing some basic functionality to the bundles running on OSGi. The XMPP service creates an Extensible Messaging and Presence Protocol (XMPP) connection and allows the bundles running on that system pass messages to other nodes. The Migration service allows the simple migration of bundles, triggered by a single method call.

3.7.4 **Block OSGi**

This is a modified version of Apache Felix provided by aicas. It provides an execution framework for bundles, offers communication through services and ensures the modularity of the components by grouping the threads of a bundle and assigning priorities and timing constraints. The real-time features offered by the modified OSGi depend on the *rt.jar* provided by the real time Java Virtual Machine.

3.7.5 **Block Transcoder**

The transcoder is written in the C language, and is based on the FFmpeg library, wrapped in a JNI wrapper, and compiled as an OSGi bundle, see [1]. The transcoding bundle is initiated by the Java Dispatcher and is provided necessary information regarding the location of the raw video and audio source, i.e., the IP Satellite Tuner. In addition, tuning parameters i.e., frequency, symbol rates, Forward
Error Correction (FEC) of the associated media stream from the satellite and associated Program Access Table (PMT), and Package Identifiers (PIDs) that are needed to gain the specific stream are abstracted from the client. A mediasstreamer client is added to the transcoder to receive a RTSP (Real-Time Streaming Protocol) live stream from a satellite source, see [10]. The source streams are in native broadcast format, i.e. as MPEG2 transport streams. The transcoder based on the requested transcoding profile, transcodes the video and audio source to a stream suitable for the recipient client device on a given TCP port. The client is redirected by the Client Controller onto this port where the transcoded content is delivered. Another OSGi transcoder can be initiated and perform the same function for a different client as required, and whether the transcoder is running on the same target hardware or on another node is abstracted from the originating client.

The actions related to the transcoder are succinctly listed in Table 3.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Transcoder</td>
<td>Being transcoding multimedia and the transcoded stream to a client.</td>
</tr>
<tr>
<td>Stop Transcoder</td>
<td>Shut down the transcoder</td>
</tr>
</tbody>
</table>

**Table 3**: A brief summary of APIs (informally described as actions) available to the Transcoder.

3.7.6 Block Real Time JVM "Jamaica"

Called Jamaica VM, this component is provided by aicas, and provides support for real-time and high performance computing. It also features a number of extensions which allow tighter control of threads, CPU usage, and memory allocation.

3.7.7 Interface to the Heuristics Module

Unlike the Heuristics Module deployed in the HPC use-case (Section 3.6.5), the Heuristics Module deployed in the Micro-Cloud use case does not handle individual jobs that run to completion once allocated. Instead, it handles a complete set of dependent jobs as a unit (i.e. a transcoder wrapped into an OSGi Bundle), and it has the possibility of migrating such units after initial allocation.

Once the Heuristics Module receives a request from the Client Controller via Dispatcher, it decides if and where to allocate the resources (i.e., a transcoder) for the new request, and whether any of the currently executing transcoders are required to be migrated.

A more detailed description of the Heuristics Module is available in Section 3.3.

Detailed References

Further details of technologies and concepts used in the micro-cloud use case are covered in deliverables D4.1 (APIs), D4.2 (Java Migration Prototype Release), and D4.5 (OS Allocation and Migration Support Final Release).
3.8 Technologies Specific to Embedded Cloud Case

The components specifically targeted to the Real Time Embedded Systems consist primarily of an ES Mapper, see Figure 19. Additionally, a Model Transformation component has been developed to changed the granularity of software description contained in the supplied Amalthea model.

The description of each component follows:

3.8.1 The Model Transformation Block

This block changes the granularity of the software described in the Amalthea model in order to speed up certain feasibility checks. Specifically, the software description contained in the Amalthea file is transformed to a relatively coarse grained software model through the following steps:

1. Sum up the time to execute a task $\tau$ by:
   - Summing up the Best Case Execution Time (BCET) of each runnable called by $\tau$ into task-level BCET, or $\tau_{BCET}$;
   - Summing up the Worst Case Execution Time (WCET) of each runnable called by $\tau$ into task-level WCET, or $\tau_{WCET}$;
   - Summing up the Average Case Execution Time (ACET) of each runnable called by $\tau$ into task-level ACET, or $\tau_{ACET}$;

2. Replace all runnables called by $\tau$ with a single runnable with timing properties $\tau_{BCET}$, $\tau_{WCET}$, and $\tau_{ACET}$. 

Figure 19: Abstract view of the components developed for micro-cloud use case. Necessary dependencies are also shown.
The transformation preserves timing properties of each task, ignoring context switching overheads. The resulting software model is considerably compact thereby significantly reducing the time it takes for the Heuristics Module to generate resource allocations. The quality of the compact software model is sufficient to make coarse-grained feasibility checks, i.e., checking whether utilization of any core exceeds 100%. Furthermore, such a transformation helps to generate resource allocations wherein a task may not be split across cores (i.e., all runnables belonging to a given task are mapped onto the same core), a scenario common to embedded systems.

### 3.8.2 Embedded System Mapper

The ES Mapper is a simple script which takes resource allocations computed by the Heuristics Module and applies it to an embedded system under test. As an example, if the objective is to simulate the embedded system on a simulator (e.g. the Timing-Architects simulator, see [12, 8]), the script maps each label to an allocated memory ³, and runnables to the allocated cores available on the processor. For details on the Timing-Architects simulator, as well as some results achieved in the context of DreamCloud, see Deliverable D6.2.

### 3.8.3 Real Time Embedded System

The Real Time Embedded System used in the context of DreamCloud activities is derived from an engine control system used in production. Therefore, the software consists of dynamics (i.e., predefined events from the environment alter the functionality of the engine control system). In line with the production version, the specification of the engine control system consists of thousands of runnables, dozens of tasks, with each task required to meet a predefined deadline. For more details on the properties of the engine control system used in the context of DreamCloud, see Deliverable D6.2.

A brief note on the evaluation platforms follows. These blocks fill in for Real Time Embedded Systems block in Figure 19. Note that the following components are not made publicly available to the DreamCloud project.

#### The Timing-Architects Simulator

The Timing-Architects simulator is the virtual prototyping tool used for evaluating the quality of resource allocations computed by the Heuristics Module, see [8, 12]. In an embedded real-time context, the model of the processor used for evaluations with the virtual platform is fixed to the Aurix 32-bit TriCore TC27x architecture by Infineon, see [2]. The execution speed of the cores (i.e., the clock rate) can vary between 100MHz and 300MHz.

The Timing-Architects simulator accepts as inputs a description of the hardware (i.e., the processor), application (i.e., the engine control system), the operating system (e.g., the OSEK, see [13]), and timing information (e.g., BCET, ACET, WCET of runnables). Such details are captured in the single Amalthea model which is an input to the Timing-Architects simulator.

The use of Timing-Architects simulator enables Bosch to quickly evaluate the performance impact due to resource allocations computed by the Heuristics Module. Compared to Timing-Architects,

---

³As of now, all labels are mapped to a global flash in order to stress the system.
extracting performance data of the same quality from actual hardware for each new comparison test would usually take about 2-3 orders of magnitude more time.

The accuracy of simulation results provided by the *Timing-Architects* has been verified in production projects executed at Bosch. For instance, in one production project, the worst-case end-to-end latencies of tasks distributed in a multicore environment as estimated by the *Timing-Architects* were within 5% of the actual observations. For the same project, the estimates of core-utilization by the *Timing-Architects* simulator was within 0.05% of the actual observations from the hardware. The *Timing-Architects* is also in use at other major automotive companies, such as Audi, Volkswagen, Continental, and the BMW group.

**The METeor Cycle Accurate Simulator**  METeor is "an interactive software development environment enabling developers to leverage the benefits of accurate and fast virtual prototypes with the visibility and control required for embedded real-time software development", see [11]. The simulator comes along with the hardware models, e.g., the Aurix TriCore processor used by Bosch for some versions of the engine control system.

More details about the evaluation of the DreamCloud toolchain on the METeor simulator will be made available as part of Deliverable D6.3.
4 Conclusions

This document described the integrated toolchain incorporating components and methods developed in the context of DreamCloud project. Specifically, methods and tools developed in the DreamCloud project are targeted to three different use-cases:

- HPC-Cloud use case.
- Micro-Cloud use case.
- Embedded-Cloud use case.

The components common to all use cases are the description of the system in the Amalthea model format, and the Heuristics Module (configured for each specific use case) which computes appropriate resource allocations.

The resource allocations are then applied to each use-case with the overall objective of:

- Extracting maximum *value* from the available system, wherein the notion of *value* is defined by the end-user (e.g. executing the application in a way which reduces its makespan increases the accrued value, also see Figure 3).
- Allocating resources to a real-time embedded application such that all tasks are able to meet their corresponding deadlines, targeted to Embedded-Cloud use-case.
- Reducing the overall energy consumption, targeted to micro- and HPC-Cloud use-cases.

A report evaluating the DreamCloud toolchain in the context of Embedded-Cloud use-case has already been completed, and is presented as Deliverable D6.2.

Corresponding evaluations for the HPC- and Micro-Cloud use cases are planned to be available as parts of Deliverable D6.4 Evaluation within Embedded Domain: Video Domain.

Overall, it is clear the methods and tools developed in the context of DreamCloud open up new possibilities that make computation more resource efficient. The approach based on reusable resource allocation heuristics that can be fine-tuned in advance with simulation and then applied to use cases using the Heuristics Module (with use case specific configuration) is the overall approach of DreamCloud.

To this end, this document shows how the different components work together towards efficient resource management in all three application domains. As a result, it is now possible to allocate resources in a way that maximizes the overall *value* that can be gained from the system, and/or reduce energy consumption. For the Embedded-Cloud use case, it is possible to map runnables and labels much faster than brute-force methods currently used in industry, significantly shortening the overall design effort.
References


