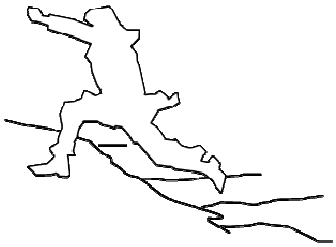


UrbanFlood

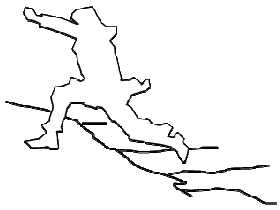
Orchestrating the (super) computing resources of the Common Information Space

Work Package 5 – D5.4

version 2.0, date April 2012

Month 24 2011


URBAN FLOOD



A project funded under the EU
Seventh Framework Programme
Theme ICT-2009.6.4a
ICT for Environmental Services and
Climate Change Adaption

Grant agreement no. 248767
Project start: December 1, 2009
Project finish: November 30, 2012

Coordinator

Urban Flood Project Office at TNO-ICT
Prof dr Robert J. Meijer

Eemsgolaan 3
PO Box 1416
9701 BK Groningen
The Netherlands

E : robert.meijer@tno.nl
T: +31 50-5857759
W: www.urbanflood.eu

DOCUMENT INFORMATION

Title	Orchestrating the (super) computing resources of the Common Information Space
Lead Author	Bartosz Balis
Contributors	Marian Bubak, Tomasz Bartynski, Marek Kasztelnik, Tomasz Gubala, Piotr Nowakowski, Grzegorz Dyk
Distribution	Public
Document Reference	UFD5.4v2.0CYF

DOCUMENT HISTORY

Date	Revision	Prepared by	Organisation	Approved by	Notes
28-11-2011	0.1	Bartosz Balis	CYFRONET		First version. Sections 1.1 and 1.2. Stub for section 1.3.
29-11-2011	0.2	Marek Kasztelnik, Tomasz Bartyński, Tomasz Gubała, Grzegorz Dyk	CYFRONET		Input for all sections
30-11-2011	1.0	Marian Bubak, Bartosz Balis	CYFRONET		Updates to all sections. Added figure of layered architecture of CIS-powered system.
30-11-2011	1.1	Piotr Nowakowski	CYFRONET		Proofreading and editing
30-11-2011	1.2	Bartosz Balis	CYFRONET		Minor corrections
15-03-2012	1.3	Marek Kasztelnik, Tomasz Bartynski, Grzegorz Dyk, Tomasz Gubala, Bartosz Balis	CYFRONET		First draft of sections about testing & validation procedures and performance measurement
26-03-2012	1.4	Tomasz Bartyński	CYFRONET		Added section on CIS and DyReAlla performance.
2-04-2012	1.5	Tomasz Bartyński	CYFRONET		Added section on DyReAlla-UFoReg communication
2-04-2012	1.6	Bartosz Balis	CYFRONET		Updated Table 4 with validation information.
2-04-2012	1.7	Tomasz Bartyński	CYFRONET		Update Table 4 with information about DyReAlla.
2-04-2012	1.8	Marek Kasztelnik, Grzegorz Dyk, Tomasz Gubała	CYFRONET		Update Table 4 with information about PlatIn, ErlMon, UfoReg, Scribe
2-04-2012	1.9	Bartosz Balis	CYFRONET		Update of Table 4, other minor updates, formatting.
3-04-2012	2.0	Piotr Nowakowski	CYFRONET		Proofreading and editing

ACKNOWLEDGEMENT

The work described in this publication was supported by the European Community's Seventh Framework Programme through the grant to the budget of the Project **UrbanFlood**, Grant Agreement no. 248767.

DISCLAIMER

This document reflects only the authors' views and not those of the European Community. This work may rely on data from sources external to the UrbanFlood project Consortium. Members of the Consortium do not accept liability for loss or damage suffered by any third party as a result of errors or inaccuracies in such data. The information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and neither the European Community nor any member of the UrbanFlood Consortium is liable for any use that may be made of the information.

© URBANFLOOD CONSORTIUM

1 Introduction

Deliverable 5.4 is the second prototype of the Common Information Space (CIS), a software framework facilitating the creation and operation of Early Warning Systems. This document summarizes the functionality of CIS developed in the second prototype (Section 2), with special focus on the orchestration of computing resources (Section 2.3). It describes an advanced implementation of the Flood Early Warning System (Flood EWS) developed using the CIS technology (Section 2.2). Also reported are the testing and validation procedures employed in the development of CIS and Flood EWS (Section 3). Finally, performance measurements of CIS are presented (Section 4).

Along with designing and developing CIS, we have also prepared an official website showcasing the CIS technology, The site can be found at http://urbanflood.cyfronet.pl. It includes an in-depth description of CIS features and its individual components, along with user and developer manuals. It is updated on a continuous basis and therefore serves as the best source of detailed information concerning CIS and any CIS-powered Early Warning Systems.

The progress of CIS development since November 2010 (the delivery date of Deliverable D5.3) can be summarized as follows:

- A new CIS service for dynamic resource orchestration (DyReAlla) has been developed and deployed.
- A new CIS service for self-monitoring of EWS software (ErlMon) has been developed and deployed.
- EWS Blueprints have been introduced, which allow us to easily create new instances of specific EWS deployments.
- The PlatIn integration platform has been improved by enabling integration with DyReAlla (registering and unregistering appliances required by EWS) and ErlMon (registering and unregistering any EWS components which need to be started on demand).
- UFoReg was fully integrated with CIS services (PlatIn, DyReAlla, ErlMon) and now serves as the metadata exchange point for the entire CIS platform.
- New components have been added to the Flood EWS (Virtual Dike Appliance and Virtual Dike Part).
- Existing components of the Flood EWS have been refined (changes include increased reliability, support for self-monitoring and dynamic resource orchestration).

Since November 2010, CIS and the CIS-powered Flood EWS have been demonstrated at various events, including the ICCS 2011 conference in Singapore (May 2011), the 2nd UrbanFlood workshop in Amsterdam (November 2011), and the Cracow Grid Workshop 2011 (November 2011).

2 The UrbanFlood Common Information Space after Year 2

2.1 CIS architecture

CIS is a software framework facilitating development, deployment and reliable operation of complex systems which rely on scientific computing, in particular Early Warning Systems for natural disasters.

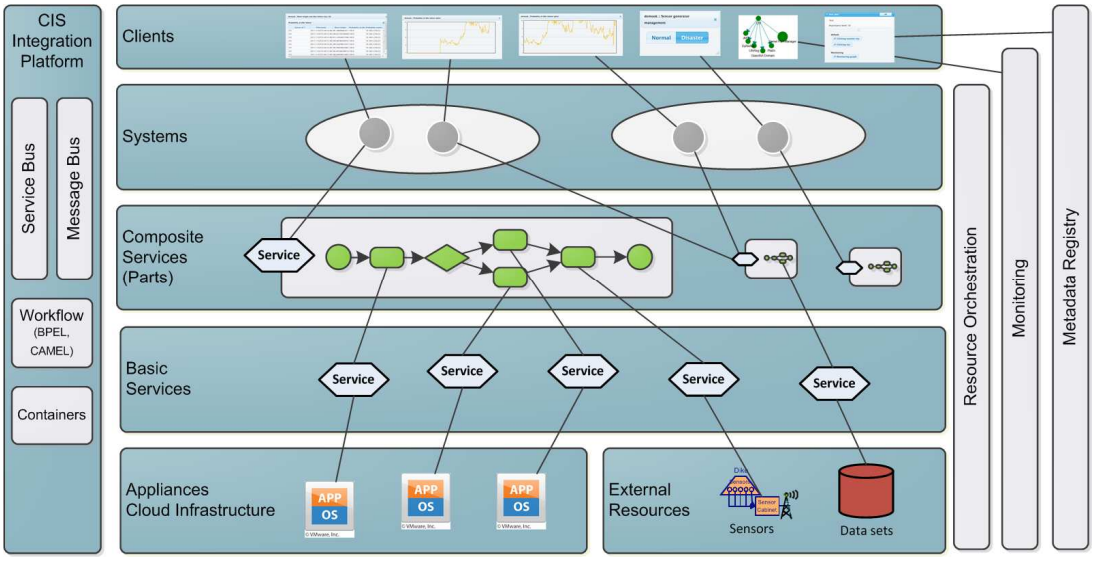


Figure 1: Layered architecture of a CIS-powered system. Resources are exposed as basic services, including scientific applications wrapped as appliances and deployed in the cloud, as well as external resources. Basic resources can be composed to provide high-level composite services. An Early Warning System is a collection of services, configured and composed in a loosely-coupled fashion.

Fig. 1 illustrates the service-oriented approach to system development as adopted by CIS. Resources (scientific applications, sensors, datasets, and others) are exposed as services. Some of these resources are managed by CIS (scientific applications wrapped as virtual images and deployed in the cloud) while others are external. Basic services can be composed to provide high-level functionality as composite services, also called system parts. A CIS-powered system is a collection of services, configured and composed in a loosely coupled fashion leveraging the message bus as a main communication medium.

Fig. 2 depicts the current internal architecture of CIS. The main components of the CIS technology stack are as follows:

- **Integration platform**: CIS core technologies for component integration, data exchange and workflow orchestration.
- Metadata registry (**UFoReg**): a generic service for hosting and querying metadata.
- Dynamic resource allocation service (**DyReAlla**): a service for dynamic allocation of resources to running Early Warning Systems.

- Self-monitoring service (**ErlMon**): provides robust software sensors for self-monitoring of CIS-based systems.

Detailed system specifications, design features and specifics of the implementation process have been presented in previous deliverables D5.1, D5.2, and D5.3, as well as in related publications [Balis2011] [Krzhizhanovskaya2011]. Up-to-date and detailed information about CIS is available on the CIS homepage (http://urbanflood.cyfronet.pl).

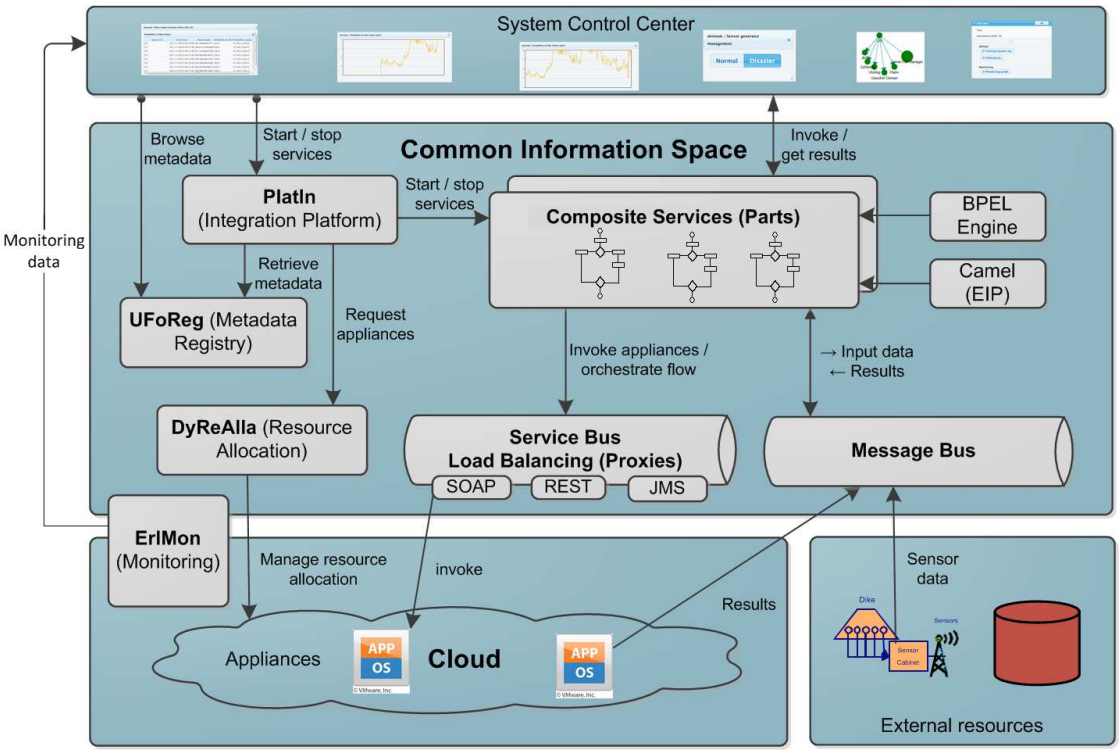


Figure 2: The architecture of the UrbanFlood Common Information Space.

The agile software development methodology has been adopted for CIS. Agile practices in use include, but are not limited to, short iteration cycles, frequent releases, test-driven development and spike solutions.

2.2 The Flood Early Warning System

The Flood Early Warning System (Flood-EWS) monitors selected sections of dikes through sensor networks and detects anomalous dike conditions. If the latter occur, alerts are generated and further inundation simulations may be performed for the purposes of prediction and damage assessment in the event of a dike failure.

Fig. 3 presents the current components of the Flood EWS, whose functionality is summarized in Table 1.

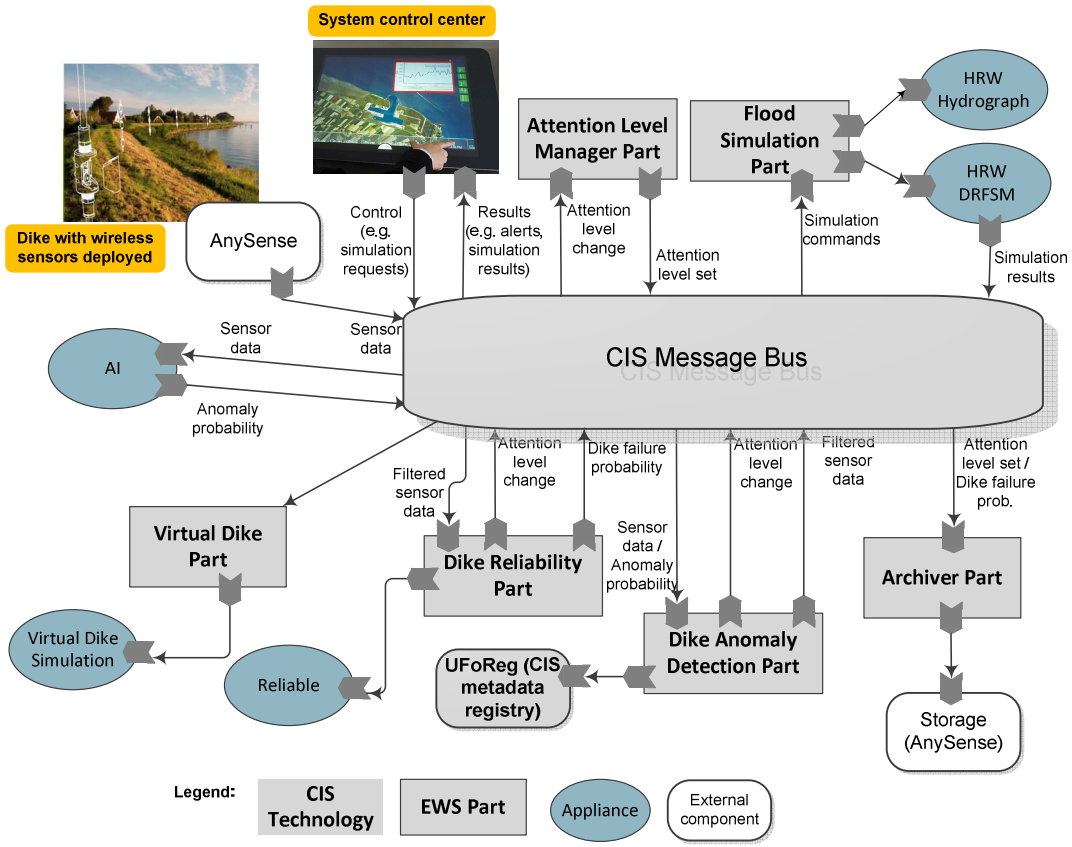


Figure 3: Implementation of the Dike Monitoring Early Warning System in the Common Information Space. The information flow through CIS is orchestrated by (1) EWS Parts and (2) CIS message bus. EWS parts are independent pieces of software developed using the CIS technology which implement high-level business logic of the EWS. Parts communicate through the message bus by publishing and consuming messages. The UFoReg metadata registry can be invoked by any component to retrieve or update metadata regarding the EWS. External components which produce or consume information may also be connected to the bus.

Table 1: Data and application components integrated in the Dike Monitoring Early Warning System.

Component name	Function	Type	Provider(s)
AnySense	Sensor data source / archive data repository	External data source	TNO
Multi-Touch Table	Visualization, user-driven steering	External GUI, client and data consumer	UvA, TNO
AI	Machine-learning based detection of anomalies in sensor signals	Appliance	Siemens
HRW Reliable	Computation of dike failure probability	Appliance	HR Wallingford
HRW Hydrograph / HRW DRFSM	Inundation simulation	Appliances	HR Wallingford

Virtual Dike (new)	Simulation of dike behaviour.	Appliance	UvA
Dike Anomaly Analysis Part	Integration of the AI appliance, generation of alert messages, filtering of sensor data	EWS Part	Cyfronet
Dike Reliability Part	Invocation of the HRW Reliable appliance, generation of alert messages.	EWS Part	Cyfronet
Flood Simulation Part	Service for performing simulation tasks specified by simulation command messages. Invokes the HRW Hydrograph and DRFSM appliances and performs necessary data transformations.	EWS Part	Cyfronet
Attention Level Manager Part	Consumes alert messages published by other parts, calculates a new alert level, and publishes the corresponding message to the message bus.	EWS Part	Cyfronet
Archiver Part	Consumes various messages published to the message bus, and passes relevant pieces of information over to the AnySense for archiving.	EWS Part	Cyfronet
Virtual Dike Part (new)	Configures simulations performed with the Virtual Dike appliance	EWS Part	Cyfronet

The UrbanFlood flagship Flood EWS is currently deployed in production mode. For demonstration purposes we have also prepared a toy EWS which can be used to try out such CIS features as:

- starting EWS,
- dynamic cloud reconfiguration based on EWS properties,
- online EWS monitoring,
- failure detection,
- EWS self-healing,
- dynamic EWS UI creation (basing on the “describe yourself” idea),
- shutting down the EWS (and freeing infrastructure resources).

Detailed description of this EWS and a video showing it in action is available on the official CIS technology webpage: http://urbanflood.cyfronet.pl/cis/doku.php?id=demo.

2.3 Resource orchestration in CIS

This section describes the new features in the Common Information Space related to the orchestration of computing resources. The Flood EWS serves as an example for describing CIS resource orchestration capabilities, which include:

- **Provisioning resources on demand**. DyReAlla is able to start existing (stopped) virtual machines or instantiate them from virtual machine templates and thus ensure that all appliances required for operation of a given Early Warning System are up and running. Basic optimization of resource allocation is in force, enabling EWS to reuse already-running appliances.

- **Acquiring external computational resources for EWS**. A proof-of-concept scenario has been developed and tested. A dedicated client for the SARA cloud (http://www.sara.nl) has been added to DyReAlla and the Virtual Dike simulation installed at SARA has been incorporated in an Early Warning System.

- **Horizontal scaling of infrastructure** (adding more instances of appliances) on the basis of EWS importance level. The optimization process involves estimations of how many instances are required for an EWS with a given importance level. Higher importance results in more resources allocated for the EWS. The importance level of an EWS can be changed while it's running – in such a case, reoptimization of resource allocation is triggered.

- **Fault tolerance based on live monitoring**. The CIS self-monitoring component maintains a graph of registered services (EWSes, EWS parts, appliances, virtual machines, CIS core components, etc.) and dependencies between them. The state of services is kept up-to-date via frequent probing. Probes use various protocols: REST (preferred), SOAP, JMX, JMS. If a given service is detected as unavailable, all dependent services are also tagged as inoperative.

- **Restarting appliances** detected by ErlMon (self-monitoring) as malfunctioning. Each appliance should provide a remote endpoint that allows querying for appliance health status. If the monitoring subsystem (ErlMon) discovers that an appliance is malfunctioning, it sends a request to DyReAlla to restart it.

- **EWS and CIS internal component status visualization**. CIS self-monitoring exposes a web-based GUI, listing the currently registered services and their status. Components are presented as a directed acyclic graph where nodes are services and edges represent dependencies between them. This interface can be seen at http://urbanflood.cyfronet.pl:9071/service.

- **Load balancing of HTTP traffic** using the industry-approved Nginx (http://wiki.nginx.org) reverse proxy. In order to support accessing HTTP-based appliances with private IP addresses the Nginx reverse proxy server is deployed on a machine with public and private network interfaces. DyReAlla registers and unregisters HTTP-based appliances in the Nginx load balancer available at a well-known URL – thus, load balancing is transparent from the point of view of EWS Parts that invoke appliances over HTTP.

- **Draw-yourself functionality** where each Early Warning System part is able to provide a dedicated user interface. The Common Information Space introduces a new component (Instance Manager UI) and UI discovery mechanisms which allow developers to create dedicated user interfaces for the EWS. All UIs contributed by EWS parts are uniformly presented them on a dedicated website. Additionally, this user interface is enriched by a mechanism which allows EWS monitoring and changing the EWS importance level. The Instance Manager UI can be found at http://urbanflood.cyfronet.pl/ui/.

- The process of creating an EWS part was streamlined by delivering EWS templates (based on the Maven archetype) with predefined dependencies and simple implementation stubs. More information about this mechanism can be found at http://urbanflood.cyfronet.pl/cis/doku.php?id=ews:partsdev:creating_camel_parts.

3 Testing and validation procedures

Proper development of the Common Information Space – a distributed system used to host Early Warning Systems – would not be possible without a dedicated testing strategy. In this section we describe our testing procedures used during development of new CIS versions. First, we show the objectives of CIS testing. This is followed by a description of the testing procedure and the tools used to test the entire system. Finally, we show our approaches to verification and validation that our system fulfills functional and nonfunctional requirements.

3.1 Objective of testing

Development of the Common Information Space relies on the Scrum methodology[1]. Development is divided into short stages (e.g. 4 weeks) – also called sprints – where the defined functionality is implemented. Following each such period we are able to provide a working prototype of CIS. To achieve this goal strong emphasis is placed on testing. In our development loop we use the following tests:

- **Unit tests** for testing small unit of code. Every unit test is executed in an isolated environment, specific to the tested feature. These tests are executed periodically by our continuum integration system (Java only).
- **Integration tests** used to test the behavior of one CIS component (PlatIn, UFoReg, ErlMon, DyReAlla) with mocked input and outputs from different components. Similarly to unit tests, these tests are executed continuously by the continuum integration system (Java only).
- **System tests** executed where snapshot versions of components are deployed into the development environment. Here, we deal with scenarios similar to integration tests but with real interactions between CIS components. These tests are usually executed manually (by interacting with the user interface or executing test scripts).
- **Regression tests** - when a bug is discovered in the system, this type of test enables us to recreate the problem. The test is subsequently added to the pool of unit or integration tests.

WP5 is not limited to developing core CIS systems but also covers parts of the Early Warning Systems. The tests used here are split into two groups:

- **Internal component tests** (unit, integration and regression tests)

[1] http://en.wikipedia.org/wiki/Scrum (development)

- **System tests** where the EWS is deployed into the working CIS and its lifecycle is subject to testing (starting/stopping/reconfiguring).

3.2 Testing procedure

The testing procedure used to develop CIS is related to the applied methodology, namely Scrum. As mentioned above, Scrum development is composed of Sprints. During each Sprint a new prototype of the software has to be produced, tested and validated. At the beginning of each Sprint we hold a meeting where we define what should be implemented during the given period. This step results in a set of so-called stories (for instance "a new EWS for monitoring dike stability will be developed"). All assignments are listed on Backlog[2]. The stories are then used to create scenarios for integration and system tests. What is more, stories are divided into more specific (development) tasks (e.g. "data from sensors needs to be gathered and stored inside AnySense"), which provide a starting points for creating unit tests.

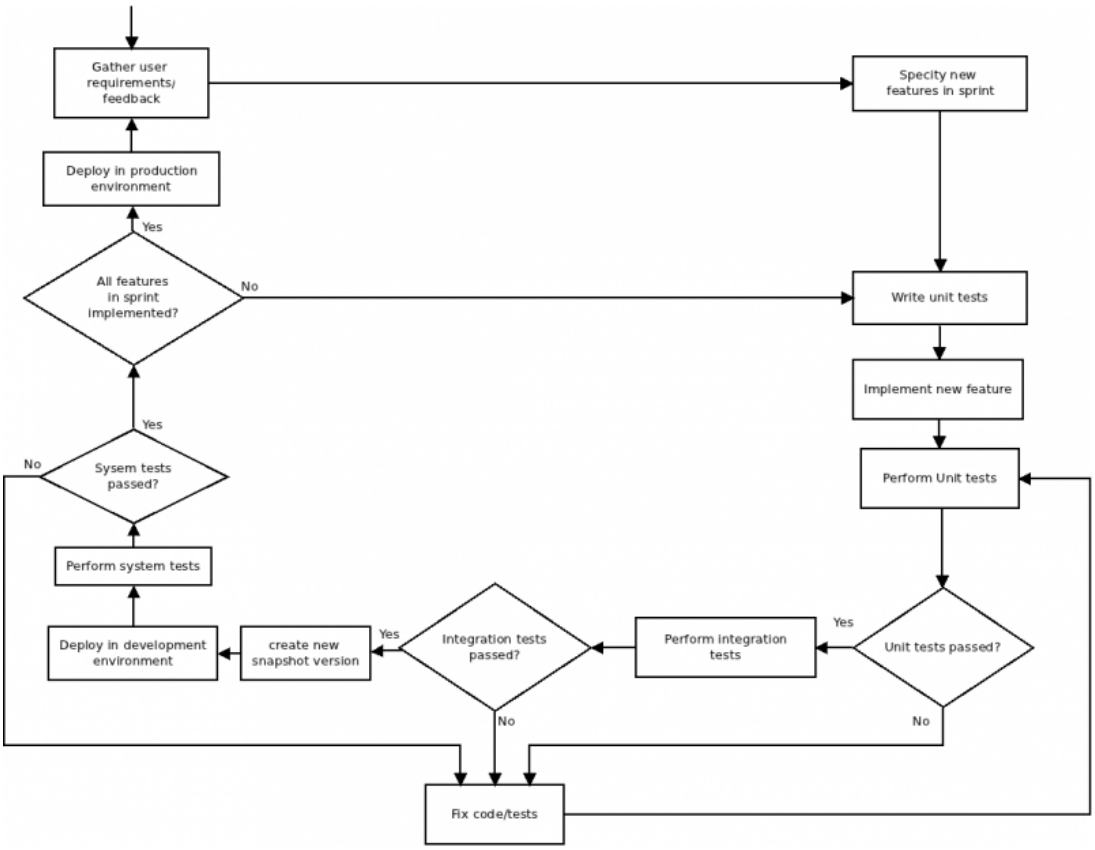


Figure 4: Testing procedure in the Common Information Space development cycle.

[2] http://chomik.cyfronet.pl/redmine/rb/master_backlog/cis

During the development process we mostly use TDD (Test Driven Development), which assumes that prior to implementation of a specific feature it is necessary to define tests for that feature. The parts of CIS which constitute the cloud subsystem follow a continuous integration methodology. All their unit and integration tests are registered inside a continuous integration system[3] which invokes tests on a hourly basis (if a new commit is available in the SVN source control system). Once a day a snapshot version of the software is built and deployed into a maven2 repository[4]. If a component test fails, an e-mail message is dispatched to the developer responsible for the affected component.

UFoReg, being a standalone server written in Ruby/Sinatra/Rack, has its own set of unit tests. Since it depends very weakly on other elements of the CIS environment, and is developed by a single person, the continuous integration principle is not applied in this case. Instead, its domain model (with many different entities from among four subdomains of UrbanFlood), its structure, consistency and validity, are tested by a set of unit tests. Moreover, the UFoReg controller layer, which provides metadata management capabilities, is tested with a separate set of unit tests. A short set of integration tests is also employed for two integration points: with PlatIn and ErlMon.

Since UFoReg includes a UI element, it also comes with a selection of browser conformance tests. These are performed manually in a periodical fashion, whenever UI views are modified. Currently the UFoReg UI works properly with Internet Explorer (ver. 9.0 running on Windows 7), Chrome (ver. 17.0 on both Linux and Windows 7), Opera (ver. 11.50 on Linux and ver. 11.61 on Windows 7) and Firefox (ver. 10.0 on Linux).

Whenever a story included in a Sprint is completed, we deploy the new CIS and new EWSes to our testing environment (http://149.156.9.56/) whose configuration mimics the production infrastructure. This enables us to run system tests. When all storied defined in a given Sprint are finished and tested the new CIS version can be released or another Sprint initiated.

It is worth mentioning that all partners of the UrbanFlood consortium are able to test new CIS and EWS release candidates using the testing environment, however this installation is not stable and some errors may occur.

3.3 Tools supporting testing

The Common Information Space is developed using three main programming languages:

- Java (DyReAlla, PlatIn, EWS hosting environment, EWSes)

[3] http://dev-gs.cyfronet.pl:8080/continuum

[4] http://dev-gs.cyfronet.pl/mvnrepo

- Ruby (UfoReg)
- Erlang (ErlMon)

Every programming language delivers a different set of tools and libraries for testing.

For Java we use:

- JUnit (http://www.junit.org/) – best know unit testing library for Java
- TestNG (http://testng.org/) very similar to JUnit but with additional features (e.g. executing the same test with different input parameters)
- <xml-unit/> (http://xmlunit.sourceforge.net/) – XML manipulation library.
- Mockup tools delivered by the Camel library (http://camel.apache.org/)

For Ruby following libraries are used:

- RUnit – Integrated standard Ruby Test Unit library
- Rack/Test – dedicated framework for REST/Webapp unit testing (part of the Rack ruby gem: http://rubygems.org/gems/rack)
- Yajl (http://rubygems.org/gems/yajl-ruby) – an efficient JSON library for REST response documents validity tests

For Erlmon we are using following testing libraries:

- EUnit (http://www.erlang.org/doc/man/eunit.html) – standard Erlang library for implementing and executing unit tests
- meck (https://github.com/eproxus/meck) – mockup library for Erlang
- Python ver. 2.7 with unittest and REST client library (http://packages.python.org/py-restclient/html/) – for implementing and executing integration tests

3.4 Overview of test results

The numbers of tests (unit and integration tests) created for each CIS component are listed in Table 2.

Table 2: Tests of CIS components.

Component name	Tests
PlatIn	14 (6 unit and 8 integration)
DyReAlla	64 (58 unit and 6 integration)
UfoReg	54
ErlMon	112 (96 unit and 16 integration)

As mentioned above WP5 also develops EWS parts. The table presented below shows the number of tests created for these parts. The number of tests is associated with the complexity of each part. For example, the *AI Archiver* has only one test as it only converts data from one format to another and sends it to AnySense. On the other hand, the *Flooding simulation* part is responsible for executing three types of simulations (Hydrograph, DRFSM and LSM) and controlling data flow between these simulations. As a result, there are 13 separate tests for this part.

Table 3: Tests of EWS parts.

EWS part name	Tests
AnySense converter	11
AI Archiver	1
Flooding simulation	13
Stability detector	11
Virtual Dike	2

CIS is a distributed system, composed of many components which frequently communicate with each other. Owing to the unit and integration tests presented above we are able to produce new versions of CIS with new features at regular intervals and streamline integration between components in the testing environment (where system tests are executed). The presented tests also speed up our development as once we define the interfaces between two modules (e.g. between PlatIn and UFoReg) we are able to create tests with mockup interfaces and develop new features without waiting for the other component to contribute the required interface. Tests created whenever a bug is discovered in our software allow us to eliminate many regression bugs (e.g. dealing with wrong data formats).

3.5 Validation

Table 4 presents the original requirements for the Common Information Space (described in Deliverable 5.2) and their fulfillment after Year 2.

Table 4: Original CIS system requirements and design decisions formulated in D5.2, and their fulfillment after Year 2.

Common Information Space requirement	Design decisions (from D5.2)	Fulfillment after Year 2

EWS should be composed of appliances	PlatIn allows creation of EWSs by connecting appliances using a supporting integration language (e.g. BPEL). DyReAlla sets up, configures and optimizes the CIS execution environment deployed on cloud resources. UFoReg stores metadata about the available appliances.	Two integration approaches have been evaluated: business process orchestration (based on BPEL) and Enterprise Application Integration (based on Apache Camel). The latter has been chosen for production for the Flood EWS.. Refer to section 2.3 for a description of resource orchestration techniques employed by DyReAlla. This metadata is available at http://urbanflood.cyfronet.pl/uforeg/configlist
EWS should use communication standards such as WS, JMS, FTP	PlatIn provides connectors for standard communication protocols. UFoReg uses REST.	Many protocols are supported including HTTP, SOAP, JMS and FTP. UFoReg provides a number of RESTful HTTP-based APIs, documented for every function ('API Help') at: http://urbanflood.cyfronet.pl/uforeg/
User should be able to set EWS priority	For every application deployed into PlatIn a priority (alert) level can be set. This information is used by EWS itself and it is routed into DyReAlla, DyReAlla allocation algorithms respect EWS priority.	For every running EWS instance an importance level (0-100) can be provided and dynamically changed. Based on the importance level resource allocation is handled by DyReAlla, e.g. the number of instances is adjusted.
CIS should deliver transformation capabilities	PlatIn delivers transformation capabilities, e.g. XSLT.	EWS Parts encapsulate transformation capabilities supporting, among others, XSLT.
CIS should be open source	All CIS components will be published as open source.	Binary packages for CIS components are available at: http://urbanflood.cyfronet.pl/cis/doku.php?id=download
User should be able to configure EWS	PlatIn uses standard protocols to configure EWSs. DyReAlla provides initial configurations for appliances at	RESTful HTTP-based APIs are provided for this purpose. The initial configurations are uploaded to appliances upon startup.

	startup time. UFoReg stores configuration.	Configurations can be accessed from http://urbanflood.cyfronet.pl/uforeg/configlist
EWS should be able to use shared cloud resources	DyReAlla manages appliances required by EWS and enables communication with appliances using standard protocols.	DyReAlla is able to provision resources required by EWS in cloud environments and release them once they are no longer needed. Appliances managed by DyReAlla can communicate with JMS bus as well as expose and invoke HTTP-based services. If appliance is using private network addressing it is registered in a reverse proxy with public address, thus it is visible for external clients.
Cloud resources should be used in effective way	DyReAlla monitors resource load and application performance to dynamically allocate resources.	DyReAlla gathers reports from cloud about load of appliances. It can also retrieve statistics about HTTP-based services from the reverse proxy and monitor JMS queue lengths. DyReAlla optimize allocation of resources by reusing already working appliances and scaling up or down the number of appliances.
CIS should be able to require resources from external providers to handle peak demands	DyReAlla can use external resource providers.	DyReAlla has a pluggable architecture and it can use any number of cloud clients that enables to use specific IaaS providers. Clients for federated Urban Flood cloud, SARA cloud and Amazon have been developed.
CIS should allow to search information about available data sources and appliances	UFoReg stores information about available data sources and appliances.	Sensor information is available at: http://urbanflood.cyfronet.pl/uforeg/sensors Appliance information is available at: http://urbanflood.cyfronet.pl/uforeg/configlist (Both can be accessed via remote APIs)
CIS should be able to host multiple EWSs and EWSs versions	PlatIn may stores different EWS versions. UFoReg may store information on multiple EWSs.	PlatIn supports execution of many EWSes basing on the information stored inside UfoReg. CIS supports the concept of EWS Blueprints which can be created by composing

		available services (EWS Parts and appliances). Currently defined EWS blueprints can be overviewed at http://urbanflood.cyfronet.pl/urbanflood.cyfronet.pl/uforeg/ewslist
CIS should support dynamic appliance deployment for ESW demand	DyReAlla can instantiate and configure appliances on demand. Depending on its priority, an appliance will be assigned the appropriate share of resources.	Refer to section 2.3 for a description of resource orchestration techniques employed by DyReAlla.
CIS should deliver sensor access capability	PlatIn allows access to sensor data using standard communication protocols. UFoReg stores sensor metadata.	PlatIn (and all EWSes) allows to access sensor data from multiple location using one of the supporting protocol (e.g. WS, REST, FTP). Sensor metadata is available at: http://urbanflood.cyfronet.plurbanflood.cyfronet.pl/uforeg/sensors
CIS should provide metadata registry allowing to describe e.g. sensor data	UFoReg can manage such data.	UFoReg provides a variety of metadata including one about sensors, cloud infrastructure, registered EWSes and running EWSes. All this information is available through internal CIS user interface and trough dedicated REST services. UFoReg also provides a batch program to populate the registry with dike and sensor metadata stored in external files. Sensor metadata is available at: http://urbanflood.cyfronet.plurbanflood.cyfronet.pl/uforeg/sensors
CIS should support EWS (self) monitoring	PlatIn delivers an API to monitor running EWSs.	Such API is provided by a separate CIS component – ErlMon. It's description can be found at http://urbanflood.cyfronet.pl/cis/doku.php?id=erlmon:api
CIS should use abstract performance parameters (e.g. CPU time, storage capacity,	DyReAlla will use VM-level metrics.	DyReAlla reads load metrics reported by Site Managers to JMS. Those metrics include CPU load, memory, disk and network usage.

memory in use) to manage cloud resources		
CIS should provide provenance tracking capabilities	PlatIn inspects running EWS and logs provenance data. UFoReg is used as a backend to store the gathered provenance data.	Provenance events are sent from PlatIn into REST endpoint exposed by separate component called Scribe. It stores data in RDF database that can be later browsed by external tools such as QuATTRo.
CIS should support secured communication channels	All CIS components support secure conversations.	UfoReg, DyReAlla, PlatIn and ErlMon secure REST communication with HTTP Base Authentication
CIS administration operations will be secured	All CIS administration operations will be secured.	All administration endpoints are secured (REST endpoints by HTTP Basic Authentication, bash operations secured by ssh username/password).
CIS should store information about component usage statistics (e.g. availability, performance)	DyReAlla will report statistics to UFoReg.	DyReAlla saves load metrics in UFoReg.
CIS should deliver easy (e.g. graphical) way of creating EWS	PlatIn delivers editors for all supported integration languages.	PlatIn is integrated with UfoReg which provides internal CIS user interface and with Multituch Table Applications (end user interface) for creating and starting EWSes. EWS parts can be created using BPEL or Camel. For these languages dedicated user interfaces are available (e.g. Netbeans for BPEL and Fuse IDE http://fusesource.com/products/fuse-ide/ for Camel).
CIS should deliver common design patterns allowing EWS creation	PlatIn delivers a set of integration languages (e.g. BPEL, Camel).	Two integration approaches are supported: business process orchestration (based on BPEL) and Enterprise Application Integration (based on Apache Camel). The latter has been chosen for production for the Flood EWS.

4 CIS and EWS Performance

This section details the performance of the Common Information Space. We focus on CIS components which are heavily loaded while starting, stopping and monitoring Early Warning Systems: DyReAlla (dynamic resource allocation component), ErlMon (EWS monitoring system) and UFoReg (registry where CIS metadata and appliance monitoring information is stored). PlatIn is used only to orchestrate "start" and "stop" requests with other components; thus performance tests for this CIS element are not necessary. We present a specification of the infrastructure currently used to host EWSes and describe the discovered bottlenecks along with possible solutions which can be used to eliminate these bottlenecks.

4.1 DyReAlla Performance

DyReAlla is responsible for assigning resources to Early Warning Systems. It obtains a specification of appliances that are required by a particular EWS as a list of required initial configuration identifiers [D5.2] and determines whether new resources should be provisioned or working appliances can be reused (i.e. shared among two or more EWSes). The decision is based on metadata describing appliances and importance levels of Early Warning Systems that use them. The importance level of a running EWS may change, causes reoptimization of allocated resources. Finally, when an EWS is terminated, DyReAlla checks which appliances can be stopped which resources can be freed. In addition, it is responsible for restarting appliances that are discovered by ErlMon as malfunctioning.

There are four different actions resulting from DyReAlla responsibilities: adding required appliances to an EWS, changing the importance level of an EWS, removing appliances due to EWS termination and restarting malfunctioning appliances. Each of these actions should be discussed separately to better illustrate the performance characteristics of the CIS platform.

Adding required appliances involves:

- Getting available resources from UFoReg for each required appliance.
- Getting appliance type (metadata used later on for optimization) for each required appliance from UFoReg.
- Optimization of resource allocation.
- Executing cloud actions like starting, stopping virtual machines, etc.
- Updating status of resource (virtual machines) affected by executed cloud actions in UFoReg.
- Registering appliances as working for particular EWS in ErlMon.
- Optionally registering appliances using HTTP protocol in proxy.

The overhead involved in adding required appliances varies a lot and depends on:

- Number of appliances.
- Number of already running EWSes.
- Number and type of cloud actions that are required (starting a stopped virtual machine is faster than instantiating it from a template which requires copying the VM image).

The following tables present an average of 10 measurements for adding the required appliances, listing the subactions involved.

Table 5: Measurements of time required to start Early Warning Systems in CIS.

Request description	Subaction	Subaction duration (in milliseconds)	Total (in milliseconds)
Starting first EWS (CIS Demo) that requires two appliances (Any Sens Message Generator and Probability Calculator). Starting 2 stopped VMs is required.	Getting available resources from UFoReg	18	454
	Getting appliance types for both required appliances	32	
	Optimization for 2 required appliances	41	
	Executing cloud actions (starting two stopped VM)	213	
	Updating status of two started appliances in UFoReg	35	
	Registering two appliances in ErlMon	31	
	Updating internal representation, coordinating other subactions	84	
Starting second EWS (CIS Demo) that also requires ASMG and Probability Calculator appliances. Virtual machines started for the first EWS can be reused so no cloud actions are required	Getting available resources from UFoReg	17	201
	Getting appliance types for both required appliances	31	
	Optimization for 2 required appliances	39	
	No cloud actions executed	0	
	No updates of status in UFoReg required	0	
	Registering two appliances in ErlMon	32	
	Updating internal	82	

	representation, coordinating other subactions		
Starting third EWS (CIS Demo) that also requires ASMG and Probability Calculator appliances. Virtual machines need to be instantiated from template	Getting available resources from UFoReg	18	35533
	Getting appliance types for both required appliances	31	
	Optimization for 2 required appliances	45	
	Executing cloud actions (instantiang two VMs from templates)	35284	
	Updating status of two started appliances in UFoReg	36	
	Registering two appliances in ErlMon	32	
	Updating internal representation, coordinating other subactions	87	

One can notice that if starting an EWS involves instantiating appliances from templates (instead of just starting stopped virtual machines), the time required to start an EWS increases dramatically. This is due to synchronous communication with the Site Manager. DyReAlla must wait until the virtual machine template is copied and an unique identifier of the instantiated template returned. The time involved in instantiating a virtual machine cannot be regarded as CIS overhead as it is introduced by the cloud layer and heavily depends on the size of the operating system and physical infrastructure (especially disk resources).

Changing the importance level of a running Early Warning System is very similar to adding required appliances. It involves the same set of subactions, but the difference is that cloud actions that are required may include stopping or destroying virtual machines because changing importance level may result in scaling up or scaling down of appliances participating in the EWS. Table 6 presents measurements which cover changes in EWS importance levels.

Table 6: Duration of an EWS importance level change request with regard to time spent on executing cloud actions.

Request description	Cloud actions	Duration of cloud actions (in milliseconds)	Total time (in milliseconds)

Changing importance level from 10 to 17 (no scaling)	none	0	45
Changing importance level from 17 to 55 (scaling up one appliance)	Instantiating one virtual machine from a template	31196	31338
Changing importance from 55 to 17 (scaling down one appliance)	Stopping one appliance	76	276

Removing required appliances is less complex than adding them or changing the EWS importance level. It involves:

- Optimization of resource allocation (discovering which appliances can be stopped).
- Executing cloud actions (stopping appliances).
- Updating status of resource (virtual machines) affected by executed cloud actions in UFoReg.
- Unregistering appliances as working for particular EWS in ErlMon.
- Optionally unregistering appliances using HTTP protocol in proxy.

The time needed to remove required appliances ranges from 25 milliseconds (if no cloud actions are performed) to 431 milliseconds (if two appliances are stopped).

Restarting an appliance is the simplest of all DyReAlla actions. When ErlMon detects a malfunctioning appliance it notifies DyReAlla which in turn checks if the appliance is registered in the HTTP proxy, unregisters it if necessary and restarts a virtual machine. Time of serving a restart request is approximately 90 milliseconds.

One can easily notice that executing cloud actions takes most of the time of serving request of any type. DyReAlla communication with other CIS components, that provides REST interfaces, is very efficient. Table 7 illustrates time spent on communication between DyReAlla and UFoReg.

Table 7: Time of communication between DyReAlla and UFoReg (average of one hundred measurements).

Request description	Time of communication [miliseconds]
Getting available resources for a given initial configuration id	24
Adding/updating description of physical host	20
Deleting a physical host	18
Adding/updating description of a virtual machine	14
Deleting a virtual machine	16
Updating load metrics of a virtual machine	11
Retrieving description of a appliance type	15

4.2 ErlMon Performance

ErlMon does not have direct impact on the performance of EWSes being lauched in the CIS environment. It works in the background, unseen by appliances or EWSes. However, there are two points where poor ErlMon behavior might affect overall system performance:

- registering EWS (done by PlatIn)
- fetching information status for a particular service (executed by graphical interfaces and DyReAlla when checking if a given virtual machine is up and running)

If ErlMon causes a delay or timeout at any of these steps, this would be immediately seen by the user.

We've conducted performance tests of ErlMon concerning its behavior in the context of these two operations. Tests were performed for 100, 200 and 300 EWSes respectively. In each case we checked:

- how long it takes to register all EWS one after another
- how long it takes to obtain the status of all EWSes with all of them being monitored.

The measured time includes REST (HTTP request-response) invocation overhead.

The testbed was an Intel Core 2 Duo 2 x 2,5 GHz machine with 4 GB of RAM. Both ErlMon and testing scripts were running on the same machine so no network delay was present. Results are presented below. All times are given in seconds.

Table 8: ErlMon performance tests.

Number of EWSes	Total time for registering	Avg. registering time	Total time for getting status	Avg. get status time
100	0,82	0,0082	0,89	0,0089
200	2,48	0,0124	1,11	0,0056
300	3,99	0,0133	5,23	0,0174

In all cases registering and obtaining the status takes less than 20 ms, which is acceptable in terms of efficient cooperation with other CIS components.

4.3 UFoReg Performance

Since a number of UFoReg performance tests were already conducted when testing DyReAlla performance, here we complete the UFoReg performance evaluation with an assessment of its throughput. Some of the elements inside UFoReg are used by human users (administrators) and, given the low number of such individuals for a single EWS, these parts are not subject to any kind of overload or scaling issues. The parts which are used by other CIS components through the HTTP API are invoked in an automated way and, as such, might impose considerable load on the registry. Here we provide results of testing the part of UFoReg which are invoked continuously and are related to cloud infrastructure monitoring, since those elements are more susceptible to traffic and scaling issues.

From the throughput evaluation perspective, the most important operation is the `'record_measurements'` API method. In the current mode of operation, the cloud monitoring elements report four measurements (disk usage, memory usage, network usage and CPU usage) for each machine in the infrastructure (there are around 20 of them in the current UrbanFlood production setup). For the time being the sampling period is set to 5 seconds, so every minute UFoReg should register 12 measurements of each quantity per virtual machine. We have decided to measure how large the cloud might grow before UFoReg cannot keep up with the **tight 5 second** schedule, forcing us to use to e.g. 10 second samples.

The test was performed on a high-end machine and with a set of parallel processes, to ensure the client side is not the real bottleneck (thus we only take into consideration the server load). Moreover, a high-throughput connection was established with the server – again, to make sure it is not the client-side Internet connection which slows down requests. The table below gives the **number of virtual machines** UFoReg is able to **register measurements** for over the period of **5 seconds**. The database state was taken from the production (to ensure that the testing environment resembles production-mode operations as closely as possible – the measurements table where insertions took place contained 8.8 million existing records, precluding any empty-db effects) and the type of hardware and setup of UFoReg were same as the production version (the only exception being one core instead of two in the production environment, making our test results more conservative – that is, the production installation is **at least** as scalable as the testbed).

The first column (parallel client processes) provides a hint as to the number of clients reporting cloud measurements UFoReg is able to handle before it saturates and slows down. Each test was performed **5 times** and the table gives the mean result (the database was reset to the same state prior to each test).

Table 9: UfoReg performance test results.

Parallel client processes	Number of served requests	Parallel client processes	Number of served requests
5	379.6	30	576.2
10	615.0	50	559.0
15	607.8	75	539.0
20	603.6	100	532.4
25	572.0		

From the results we clearly see that with approximately 10 client processes we were able to saturate the UFoReg server capabilities, with the peak throughput value of 615 virtual machines handled in the **5 seconds** time slot. However, even when the number of client processes is high UFoReg retains good vertical scaling, lowering the number of handled measurements by around **10%** for as much as a hundred different client processes reporting monitoring data (this is also due to the excellent underlying Apache and Passenger load balancing capabilities). The most important conclusion from our test is that in its current setup UFoReg is able to handle more than 500 running, active virtual machines in the cloud infrastructure before further scaling actions have to be taken (e.g., *sharding* the internal MongoDB database, splitting the frontend UFoReg server into two or simply deploying it on more CPU cores). We believe that this performance is satisfactory given the current scale of UrbanFlood operations (around 20-30 virtual machines).

4.4 Cloud infrastructure

The UrbanFlood cloud is powered by the following hardware resources:

Compute resources used to run Virtual Machines are composed of 2 classes of servers:

- A single standard 1U rack server equiped with 2 Intel Xeon 5150 CPUs (2.66 GHz, dual-core), 4 GB of RAM and 320 GB of local HDD (SATA, Western Digital Caviar WD3200YS)
- 4 blade servers in dual-server configuration: HP ProLiant BL2x220c G5 equiped with 2 Intel Xeon L5420 (2.5 GHz, quad-core), 16 GB of RAM and 120 GB of local HDD (SATA, HP GJ0120CAGSP) each.

All mentioned servers are connected using 1 Gigabit Ethernet interconnect.

In addition to the mentioned computing resources 1 TB of storage is provided on an external Hitachi disk array, using 15 kRPM SAS drives in RAID 6 configuation. Those resources are

delivered to the machines using iSCSI based Storage Area Network (SAN) and shared between all servers in a single pool.

The XenServer software (including Xen Hypervisor) is used as the virtualization stack. The rack server mentioned above runs the 5.5.0 version of the software and all blade servers, which are grouped into a single Resource Pool, are running version 5.6.0. Both pools use private IP adresses for management, with an OpenVPN tunnel, and public IP addresses for VMs running in the cloud (dynamically assigned by an external DHCP server).

4.5 Discovered bottlenecks

While developing and hosting EWSes on CIS we discovered two bottlenecks:

- The first bottleneck is connected with the communications layer used by EWSes: JMS (http://en.wikipedia.org/wiki/Java_Message_Service). A large volume of data is sent using this channel (e.g. data from sensors, input and output messages from and to simulations). Hopefully this problem can be addressed by mechanisms delivered by the JMS implementation currently in use, namely ActiveMQ Clustering (http://activemq.apache.org/clustering.html).
- The second issue is connected with the limited amount of computing power we currently have in UrbanFlood. This problem can be solved by procuring additional resources from external vendors (e.g. Amazon).

Over the last year of the project we plan to perform extended scalability tests. Results of these tests will be published in the final WP5 deliverable.

5 Summary

During Year 2 of the Project, WP5 has made major progress on the development of the Common Information Space and CIS-based Flood Early Warning Systems. Components added include DyReAlla (a dynamic resource allocation service) and ErlMon (a system for self-monitoring of the CIS and EWS software components). Existing CIS components (the PlatIn integration platform and the UFoReg registry) have been substantially extended. CIS is now capable of multi-faceted management of computational resources for the purpose of running Early Warning Systems. Thus, milestone MS7 has been successfully achieved.

References

[Balis2011] B. Balis, M. Kasztelnik, M. Bubak, T. Bartynski, T. Gubala, P. Nowakowski, and J. Broekhuijsen. The UrbanFlood Common Information Space for Early Warning Systems. *Procedia Computer Science*, 4:96-105, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.

[D5.2] UrbanFlood D5.2 deliverable "Specification of the architecture and interfaces of the Common Information Space"

[Krzhizhanovskaya2011] V.V. Krzhizhanovskaya, G.S. Shirshov, N.B. Melnikova, R.G. Belleman, F.I. Rusadi, B.J. Broekhuijsen, B.P. Gouldby, J. Lhomme, B. Balis, M. Bubak, A.L. Pyayt, I.I. Mokhov, A.V. Ozhigin, B. Lang, and R.J. Meijer. Flood early warning system: design, implementation and computational modules. *Procedia Computer Science*, 4:106-115, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.