FP7-SMARTCITIES-2013
Project number: 609062
http://www.smartie-project.eu/

SMARTIE

Deliverable D4.3

System and Management Mechanisms

Editor:	Martin Bauer (NEC)
Dissemination level: (Confidentiality)	PU
Suggested readers:	Consortium/Experts/other reader groups
Version:	0.9
Total number of pages:	38
Keywords:	Large scale smart city platform mechanisms, processing flow optimization, discovery, retrieval

Abstract

In the SMARTIE deliverable D2.3, different architectural configurations have been discussed that range from constrained devices to non-constrained IoT Servies and large scale IoT deployments. This deliverable describes the work conducted in T4.2 on the design of service management for discovery and retrieval that primarily targets the case of large scale IoT deployments for the Smart City. This also relates to the Smart City Information Centre use case envisioned in deliverable D2.1 for which a distributed large scale service management is required.

The main focus of this deliverable is on processing flow optimization and related functionality. The underlying idea is that information originating from IoT sources, in particular sensors, flows through the distributed system and is processed in suitable places. Processing flow optimization optimizes the placement of the processing in the distributed system according to an optimization function and taking into account constraints, especially security constraints. As a basis for processing flow optimization, the discovery of sources and processing nodes is required and the plan resulting from processing flow optimization that maps processing units to processing nodes also needs to be executed, i.e. processing units need to be dynamically deployed and the flows have to be initiated.

As part of WP5, processing flow optimization will be integrated with the SMARTIE components RD (resource directory) enabling discovery and the Capability Manager for authentication and authorization.

Disclaimer

The information, documentation and figures available in this deliverable are written by the SMARTIE partners under EC co-financing (project number: 609062) and does not necessarily reflect the view of the European Commission.

Impressum

[Full project title] Secure and sMArter ciTIes data management

[Short project title] SMARTIE

[Number and title of work-package] WP4
[Document title] System and Management Mechanisms
[Editor: Name, company] Martin Bauer, NEC

[Work-package leader: Name, company] Antonio Skarmeta, UMU

Copyright notice

Executive Summary

In the SMARTIE deliverable D2.3, different architectural configurations have been discussed that range from constrained devices to non-constrained IoT Servies and large scale IoT deployments. This deliverable describes the work conducted in T4.2 on the design of service management for discovery and retrieval that primarily targets the case of large scale IoT deployments for the Smart City. This also relates to the Smart City Information Centre use case envisioned in deliverable D2.1 for which a distributed large scale service management is required.

The main focus of this deliverable is on processing flow optimization and related functionality. The underlying idea is that information originating from IoT sources, in particular sensors, flows through the distributed system and is processed in suitable places.

Processing flow optimization optimizes the placement of the processing in the distributed system according to an optimization function and taking into account constraints, especially security constraints. As a basis for processing flow optimization, the discovery of sources and processing nodes is required and the plan resulting from processing flow optimization that maps processing units to processing nodes also needs to be executed, i.e. processing units need to be dynamically deployed and the flows have to be initiated.

The following required components are described in this deliverable:

- Discovery

  The Discovery component enables applications and system components to find or look up IoT resources known by the system at run-time. IoT resources include sources of information, but also processing nodes on which processing units can be deployed. The following discovery component instances are described in detail:

  - RD (Resource Directory)
  - FIWARE IoT Discovery supporting Geographic Scopes
  - Network Information Directory

- Processing Flow Optimization (PFO)

  The Processing Flow Optimization component maps an abstract task topology to a concrete task topology that can be deployed. The abstract task topology describes how information from information sources should flow to processors for processing and ultimately to the receiver of information.

- Processing Unit Deployment (PUD)

  The Processing Unit Deployment component is responsible for deploying processing units to processing nodes according to the concrete task plan provided by the Processing Flow Optimization and initializing the flow of information.

- Processing Unit Storage

  The Processing Unit Storage stores the processing unit executables to be deployed onto processing nodes by the Processing Unit Deployment.

Furthermore, the interactions between the components, as well as how to secure them in the context of SMARTIE is described. For the latter, the focus is on the integration with the Capability Manager to enable authentication and authorization.

As part of WP5, processing flow optimization will be integrated with the SMARTIE components RD (resource directory) enabling discovery and the Capability Manager.

List of authors

Company	Author
NEC	Martin Bauer, Bin Cheng, Apostolos Papageorgiou
DNET	Boris Pokrić, Stevan Jokić
UMU	Alfredo Quesada Sánchez, José Luis Hernández Ramos

Table of Contents

List of Figures

Abbreviations

AES	Advanced Encryption Standard
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
DCapBAC	Distributed Capability-Based Access Control
Dev	Device
GW	Gateway
IoT	Internet of Things
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
NID	Network Information Directory
NP	Nondeterministic Polynomial time
OSGi	OSGi (formerly Open Services Gateway initiative**)**
PDP	Policy Decision Point
PFO	Processing Flow Optimization
PUD	Processing Unit Deployment
RD	Resource Directory
RDNode	Resource Directory Node
SMARTIE	Secure and sMArter ciTIes data management
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1 Introduction

To realize the vision of a smart *city*, a large number of systems need to be integrated, so we are actually talking about a massive distributed system of systems. The scale and dynamics of such a system require that applications are supported by platform mechanisms that enable the run-time discovery of resources as well as the flexible processing of information within the system that enables the efficient extraction of the information needed by the application

This deliverable describes components for a large scale smart city system with a focus on flexible support for distributed processing, in particular on how to optimize the placement of processing within the system according to optimization criteria expressed by an optimization function. Furthermore, additional constraints regarding the placement need to be taken into account, in particular regarding security aspects. A key concept in this respects are the security domains, which allow the requester to specify that certain processing tasks have to be executed within the domain as the nodes in this domain are trusted.

The proposed components for the optimized planning and setting up of processing flows are integrated with relevant SMARTIE components, in particular the RD, which is responsible for storing the information about resources, e.g. sensor resources, but also processing nodes on which processing units can be deployed. As the described components assume a large scale, highly distributed system for a smart city as described in [2], there is no direct integration with the SmartData platform of Portugal Telecom that has a more centralized architecture regarding the processing of information.

1.1 Relation to S&T Objectives

This deliverable primarily addresses the following S&T Objectives.

> O3 – Develop new technologies for trusted information creation and secure storage for the information service layer

The presented solution takes trust-related constraints into account when deciding where to process information and thus create new information.

> O4 – Develop new technologies for information retrieval and processing guided by access control policies in the application layer

The focus of the deliverable is on optimizing the processing and retrieving of information, taking into account constraints, especially security-related constraints. The solution is especially targeted at large scale distributed system of systems as they can be found in Smart Cities. Here, available information sources change dynamically, requiring efficient discovery of information and a flexible system for processing information (aggregation, filtering, transforming interpretation …). To ensure that access control policies are adhered to, the required interactions with the Capability Manager are described.

1.2 Beyond State-of-the-Art Contributions

In this deliverable we propose an architecture that allows the dynamic deployment of processing nodes to heterogeneous compute nodes ranging from sensor nodes to gateways, edge servers and cloud nodes. Existing work has primarily looked at more homogeneous cases, e.g. only in the cloud or only within a sensor network.

In general, the optimization problem is a hard one that has, for example, been tackled by focusing on a particular optimization function and using the A* algorithm or by using a heuristics that takes into account the communication characteristics of the tasks. We do not want to limit ourselves to one particular optimization function and, unlike typical existing cases, we have a system consisting of more heterogeneous nodes. Finally, in our case it is important to take into account (security) constraints and we expect that this will significantly reduce the solution space.

Thus, we propose to tackle the hard optimization problem by using a two-step mapping approach. Since the types of nodes to which the processing tasks can be mapped are rather heterogeneous, e.g. sensor nodes, gateways, edge servers or cloud nodes, we first combine similar nodes into "power nodes" and do the mapping on this basis and only in a second step we map the processing tasks assigned to the power nodes to actual nodes

Also, we have shown how the processing flow optimization, deployment and execution can be secured using the SMARTIE Capability Manager for authentication and authorization.

2 Overview of System and Management Mechanisms

This section gives an overview of the system and management mechanisms developed in Task 4.2 of the SMARTIE project. The focus is on mechanisms needed for a large scale smart city platform that goes beyond the actual deployments developed for WP6.

2.1 High-level Architecture

The architecture considered here is based on "Scenario 3 – Large scale IoT Deployment with Virtual Entity level services" as identified in Section 3.2 of SMARTIE Deliverable D2.3 – Initial Architecture Specification [2] that targets a large scale distributed Smart City deployment with multiple players involved. As identified there, the following components are required to cover such a large scale IoT scenario: communication, discovery, processing, authentication/authorization and event detection. As already identified in [2], off-the-shelf secure communication mechanisms can be used, so there is no need for in-depth coverage of this topic in this deliverable. Event detection has been covered in detail in D4.2 [3], so the interested reader is referred there.

In the context of this deliverable, discovery refers to finding the relevant resources and information that are already known within the IoT system. For a large scale IoT system the available resources and information may dynamically change, so having an a-priori fixed configuration does not make sense. Rather, applications or system components will dynamically look for resources that are suitable for the desired purpose. The RD as the component for discovery and look-up chosen in SMARTIE is described in Section 3, together with other alternatives.

The main focus of this deliverable will be on how to optimally support the processing of information in a distributed environment. The assumption is that there are different places where this processing can take place, e.g. on the sensor node, on the gateway, on a specialized server or on a cloud node. For the placement of processing different optimization criteria can be defined, taking into account the sources of information, the communication links, the processing power of the nodes etc. In addition, especially from a security point of view, constraints need to be taken into account, e.g. that raw information may not leave a certain domain, the availability of encryption mechanisms and the level of trust in the respective node doing the processing.

To ensure that only authorized users can process and receive certain information, user applications request the needed tokens from the Capability Manager, which will then be checked by the requested component before executing the request. The interactions with the Capability Manager will be described in Section 6.3, a more detailed description of the Capability Manager itself can be found in [10].

The following subsections provide a short description of each of the high-level architectural components. A typical interaction between the components is shown in Section 2.2.

2.1.1 Discovery

The Discovery component enables applications and system components to find or look up IoT resources known by the system at run-time. IoT resources include sources of information, but also processing nodes on which processing units can be deployed.

2.1.2 Network Information Directory (NID)

The Network Information Directory provides information about how the nodes in the network are connected and the relevant properties of these connections, e.g. their bandwidth and latency.

2.1.3 Processing Flow Optimization (PFO)

The Processing Flow Optimization component maps an abstract task topology to a concrete task topology that can be deployed. The abstract task topology describes how information from information sources should flow to processors for processing and ultimately to the receiver of information.

2.1.4 Processing Unit Deployment (PUD)

The Processing Unit Deployment component is responsible for deploying processing units to processing nodes according to the concrete task plan provided by the Processing Flow Optimization and initializing the flow of information.

2.1.5 Processing Unit Storage

The Processing Unit Storage stores the processing unit executables to be deployed onto processing nodes by the Processing Unit Deployment.

2.1.6 Capability Manager

The Capability Manager is responsible for authenticating the user and authorizing a request. If the authenticated user is allowed to make the request, the Capability Manager issues a cryptographically signed token that is passed on with the actual request. The recipient will check the token and authorize the request.

2.2 High-level Interactions between Components

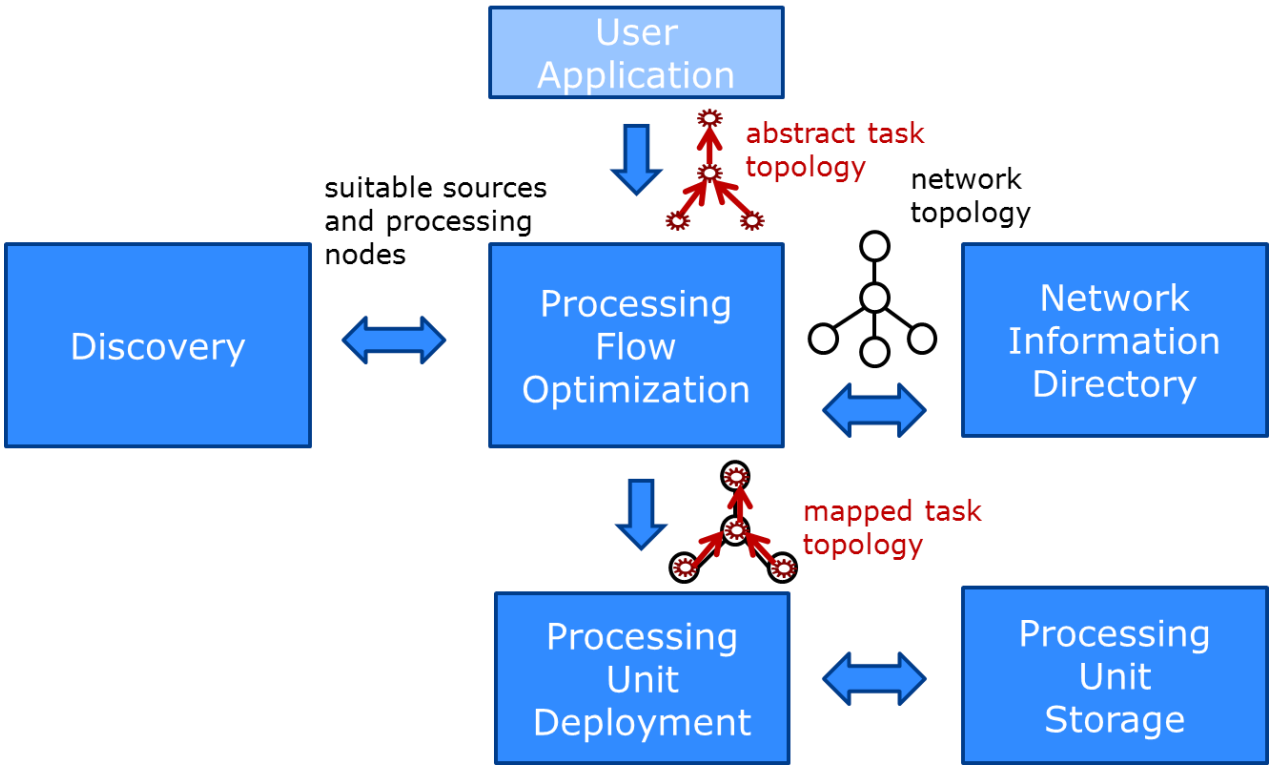

Figure 1 High-level Component Interactions

Figure 1 shows the high-level interactions between the functional components that are needed to set up processing flows according to the specification provided by the user application. The User Application provides this specification in the form of an abstract task topology to the Processing Flow Optimization component, giving also constraints that have to be taken into account for the solution. The abstract task topology specifies the required sources as well as the intermediate processing steps that have to be mapped to concrete source and processing nodes and the required flows between sources and processing nodes. The Processing Flow Optimization requests suitable sources and processing nodes from the Discovery. If specific sources have been specified, this is a simple look-up operation, whereas if only a description of suitable source has been specified, a discovery operation is needed. In the same way, candidate processing nodes need to be discovered. Finally, the Processing Flow Optimization component needs the network topology between the (candidate) sources and candidate processing nodes, which is retrieved from the Network Information Directory. Then the Processing Flow Optimization maps the abstract task graph onto actual

sources and processing nodes using an optimization function and taking into account the constraints provided by the User Application. The resulting mapped task graph is then handed over to the Processing Unit Deployment that deploys the processing units on the chosen processing nodes, configures them and starts the flows from the sources through the processing units to the sink identified by the requesting User Application.

3 Discovery

In a large-scale IoT system, e.g. for a complete smart city, it is important to support the discovery of resources and, ultimately, information. It cannot be assumed that applications are explicitly configured for all the resources in the system and that such a configuration is adapted each time the availability of resources changes. Applications, but also advanced system mechanisms, like processing flow optimization, need to be able to discover resources that can provide information, process information or enable actuation.

In SMARTIE, the Resource Directory (RD) has been chosen as the component for discovering and looking up IoT resources like sensor, actuator or processing resources. It was therefore decided to integrate Processing Flow Optimization with the RD. Thus, in this deliverable, we describe the RD, its functionalities and how IoT resources are modelled, giving examples how sensor resources and compute nodes for processing are modelled in SMARTIE. In addition, an approach for discovery is described that supports geographic scopes. Finally a short description is given describing the Network Information Directory (NID).

3.1 Requirements on Discovery

It must be possible to model and store descriptions of IoT resources including a unique identifier, what information they can provide, how this information can be accessed and additional meta information including the location for which the information is provided.

It must be possible to model and store descriptions of special IoT resources that enable the dynamic deployment of processing units, including meta information about the processing capacity.

It must be possible to look up IoT resources based on its unique identifier.

It must be possible to discover IoT resources based on the information they can provide and relevant meta information.

3.2 Resource Directory (RD)

The RD provides persistent storage of resources. Resources are described using human/machine readable resource description format. The RD is based on the usage of a free form of tree based resource description. The resource description model is shown in Figure 2. RDNode is the key element in the resource description building. RDNode caries node data like name/value pair as well as parent/children links needed to build a tree based structure of the resource description. Standard types are defined for resource node values like String/Numeric and Container which contains RDNodes as children.

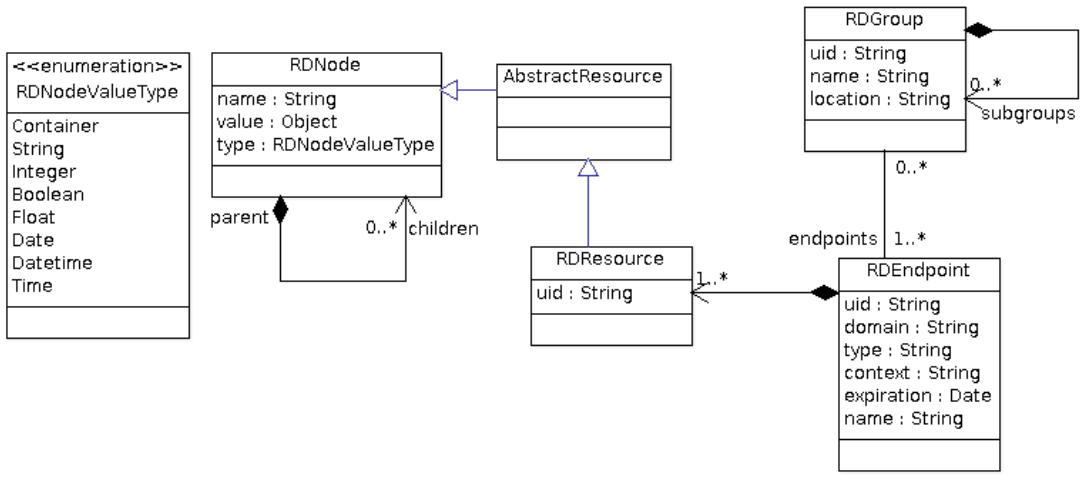

Figure 2. Resource description data structure model

Example of temperature sensor which provides temperature in Celsius in the JSON format.

```
name = resource-description
      type = Container
      encrypted = false
      value = null
            name = descriptor
            type = Container
            encrypted = false
            value = null
                  name = type
                  type = String
                  encrypted = false
                  value = temperature

                  name = unit
                  type = String
                  encrypted = false
                  value = celsius

                  name = response
                  type = String
                  encrypted = false
                  value = JSON

                  name = name
                  type = String
                  encrypted = false
                  value = t

                  name = endpoint-url
                  type = String
                  encrypted = false
                  value =
http://srv.dunavnet.eu/rep/rest/data/latest/compact/path/eb355255040956196.t
```

Example of processing node description in JSON format:

```
name = resource-description
      type = Container
      encrypted = false
      value = null
            name = myProcessingNode4
            type = Container
```



```
encrypted = false
value = null
      name = type
      type = String
      encrypted = false
      value = processing_node

      name = owner
      type = String
      encrypted = false
      value = NEC-HD

      name = memory
      type = String
      encrypted = false
      value = 4GB

      name = MIPS
      type = Integer
      encrypted = false
      value =  100000

      name = security_domain
      type = String
      encrypted = false
      value =  domainB, domainA, domainF
```

The RD provides REST interfaces over HTTP and CoAP. The HTTP services support XML resource description serialization. The CoAP interface uses the CoRE link [11] format for resource description serialization.

Resources may be listed using the HTTP GET method in the following URL format: <RD_HOST>/rd

Access to the specific resource description is available using the following URL format: <RD_HOST>/rd/<RESOURCE_UID>

Allowed HTTP methods are GET, POST (for resource creation), PUT (for resource updating) and DELETE (for resource deleting).

RD querying is provided through the RLI interface. Querying is performed using HTTP GET on the following URL format: <RD_HOST>/rli?<QUERY>

QUERY is encoded as URL parameters.

Example: <RD_HOST>/rli?sensor.type=temperature will return resources with sensor type temperature.

Analogue interfaces are available on the CoAP interface.

The RD access control is implemented using the DCapBAC SMARTIE component. Figure 3 illustrates the ekoBus device sequence diagram registering a device in the RD using SMARTIE components to obtain appropriate capability tokens and then to perform device registration in the RD.

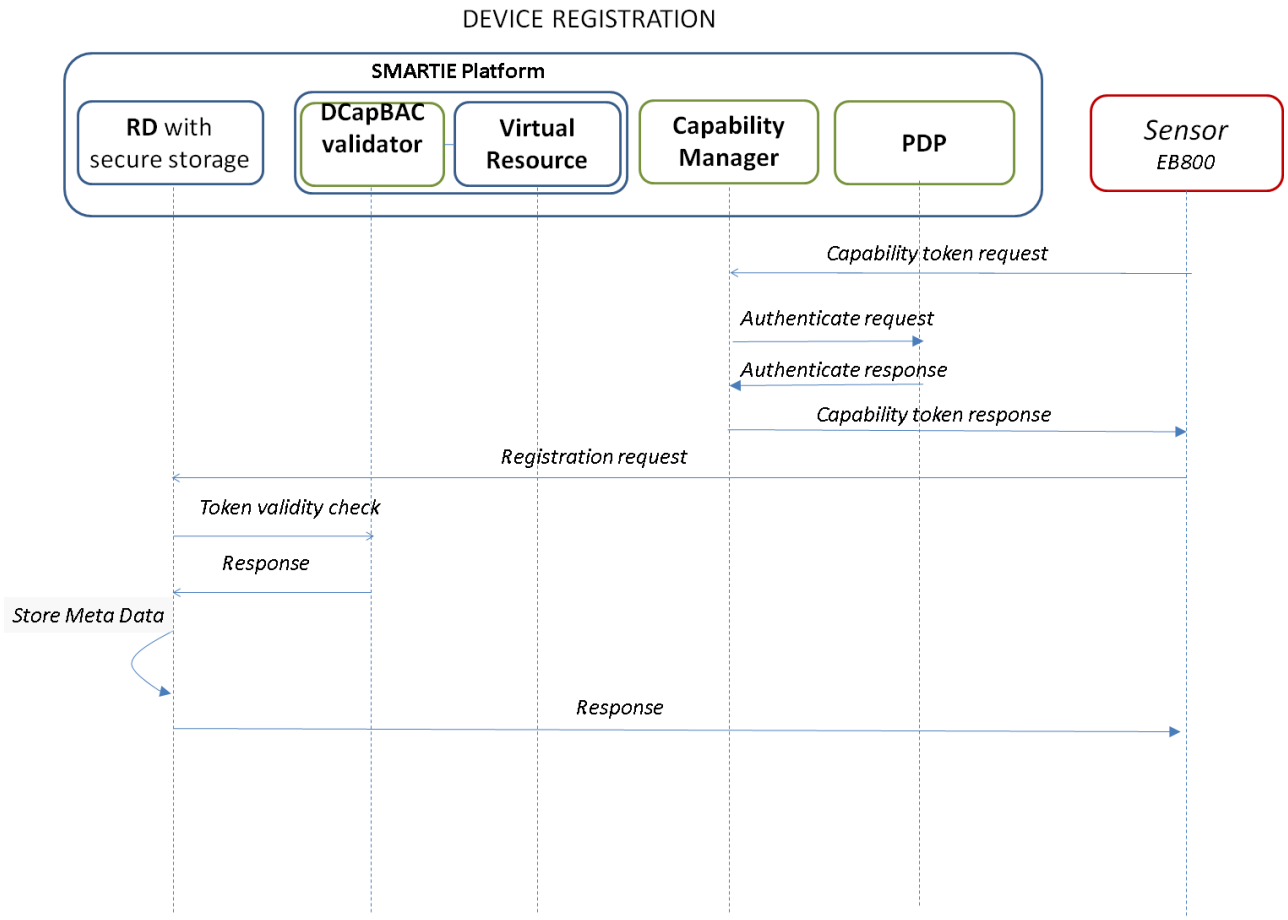

Figure 3. ekoBus device registration in the RD

The Capability token should be placed in the HTTP header of the request to perform authentication on HTTP interfaces. The Capability token should be encoded as Base64 to ensure a valid HTTP header structure [12]. The RD checks presence and validity of the capability token and adapts its response according to the result of the validity check. The RD will send appropriate cookies if the authentication was successful. If the user application supports cookies it may be used for following accesses to the RD.

Part of the resource description may be encrypted. Authorized users of the RD receive the response with decrypted resource descriptions. Unauthorized access will provide a response with encrypted values.

Encryption in the RD is performed using JAVA Security package for AES [13].

In the previous resource description example, the end point URL-part of the resource description is protected, thus unauthorized access to the RD will provide an encrypted end point URL in the response:

```
<resource-description>
   <uid>002</uid>
   <created>2015-10-16 15:11:27</created>
   <updated>2015-10-16 15:11:27</updated>
   <descriptor>
      <type>temperature</type>
      <unit>celsius</unit>
      <response>JSON</response>
      <name>t</name>
      <endpoint-url encrypted = "true">
```



```
QBkCYonhgFm7q6OKVxhwX4p3dUHz48YqfqUo7Hq8k5fnUo1zbP7P2QCbbg3KAKcqDkBGX5tptzMpqgHb
pFFFv5Po+Z6eKRY6uu4boMpGyoM7yKThsZsEhbuWUWaUkN6u
</endpoint-url>
   </descriptor>
</resource-description>
```

Authorized accesses to the RD will provide response with a decrypted end point URL.

The RD maintains “created” and “updated” fields of the resource description.

3.3 FIWARE IoT Discovery supporting Geographic Scopes

The IoT Discovery General Enabler in the FIWARE platform is responsible for enabling the discovery data sources that can provide the requested context information. For that purpose it implements the NGSI-9 context management interface [14].

NGSI-9 allows the registration of context sources providing a geographic extent specifying the area for which they can provide the information.

For discovering context sources, applications can specify a geographic scope which is then matched against the geographic extent registered by sources. In case of an overlap, the context source will be included in the result set (provided it also fits any other restrictions that have been specified).

The geographic shapes supported are point (coordinate with latitude, longitude), rectangle (NW and SE coordinates with latitude and longitude), polygon (list of coordinates with the last one being the same as the first to close the polygon) and circle (centre coordinate and radius).

Figure 4 Discovery based on geographic scopeFigure 4 shows an example of a geographic discovery. The registered context sources cover the areas 1A, 1B, 2 and 3. The scope of the discovery request is shown by the red dashed line. In the given case, the extents of context sources 1A, 2 and 3 intersect with the geographic scope and would be returned within the result set.

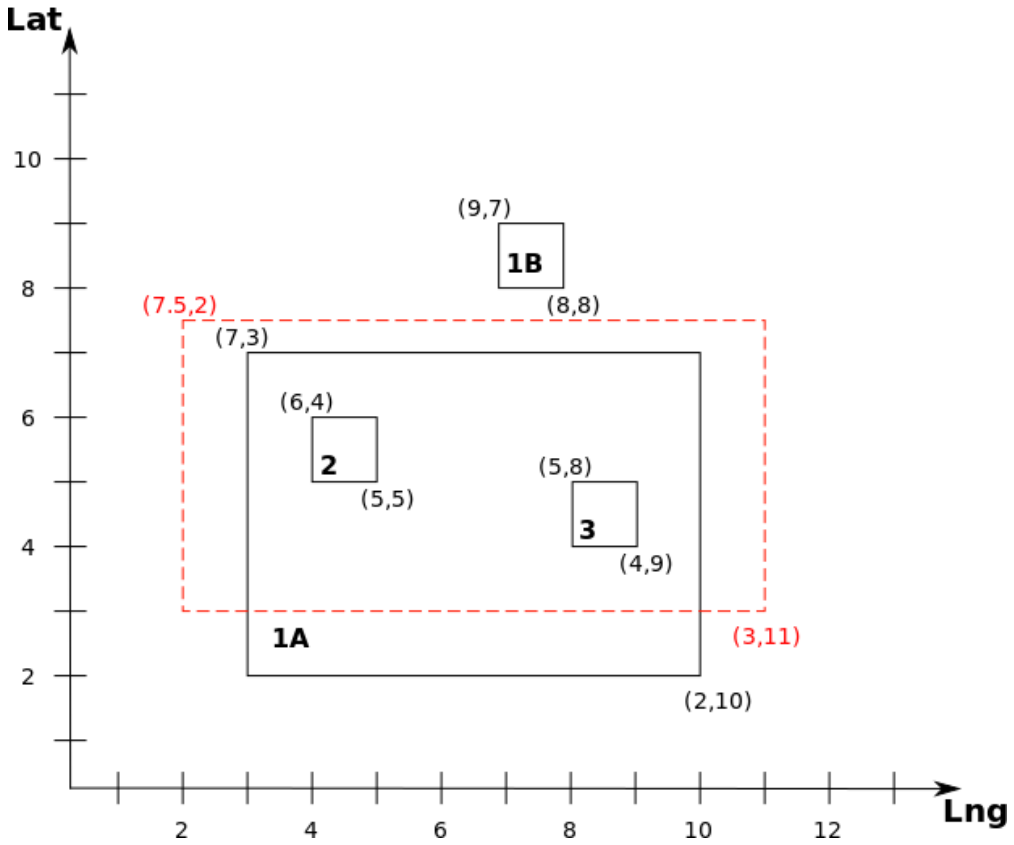

Figure 4 Discovery based on geographic scope

In addition to a one-time discovery, applications can also subscribe for fitting context sources, i.e. they will be notified about new context sources becoming available and other context sources no longer being available.

Discovery with geographic scopes can be very useful in smart city use cases as the same application can easily be used for different areas without having to explicitly configure it beforehand. Also, due to the large scale and thus the large number of potential sources, the available context sources will change dynamically. For the processing flows, monitoring these changes and adapting the processing flows accordingly will be an important feature, but is beyond the scope of the SMARTIE project.

3.4 Network Information Directory

The Network Information Directory is a component that based on a set of nodes given as input can provide the network topology with the relevant properties that are needed for processing flow optimization, e.g. bandwidth and delay. As this component is not the focus of the work in SMARTIE, only a limited prototype will be used that provides the compute node topology annotated with static properties, so that the information for processing flow optimization is available.

4 Processing Flow Optimization

A Processing Flow task topology describes how information is processed in the IoT system, i.e. what sources are needed, how the processing components process and integrate information from different sources, and which component ultimately consumes the information. The abstract processing flow topology describes *what* has to be done, but not *where*. Processing flow optimization is about mapping the abstract processing flow onto actual system resources, so that in a following step the processing components can be deployed and finally the flow of information can be triggered. When mapping the processing flows onto system resources, constraints, especially security constraints, have to be considered, restricting the possible solutions for the optimization step.

4.1 Motivation

In large-scale smart city scenarios, the processing of information flows is required; different processing steps may be needed to derive the desired information.

Due to limitations in available bandwidth, latency requirements, but also security constraints, processing cannot only be done in a centralized cloud infrastructure. Instead, distributed processing is required. Different processing steps may be done on different nodes in a heterogeneous IoT network, e.g. on sensor nodes, gateways, dedicated servers or a cloud infrastructure. The processing units required for the different steps need to be deployed on the most suitable nodes.

The main focus of this work is to map the processing units to nodes taking into account (security) constraints and optimize according to an optimization function.

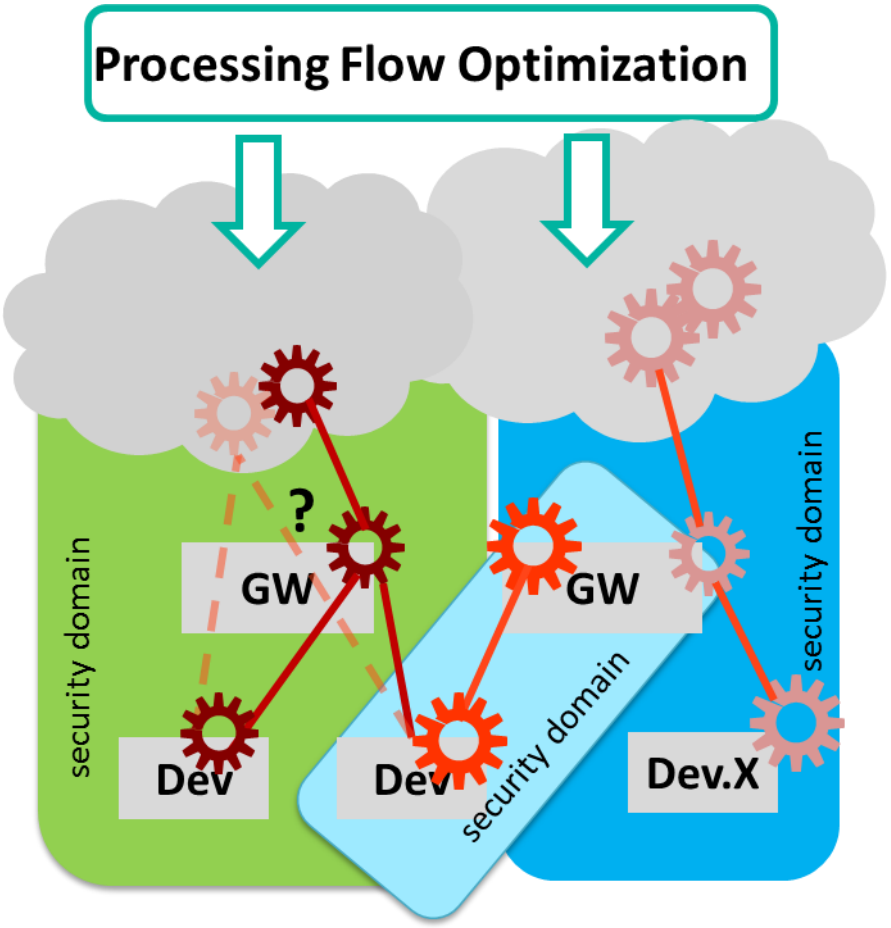

Figure 5 Example of processing flow optimization with security domains

Figure 5 sketches an example with different processing locations, i.e. devices, gateways and clouds. As shown, they can be in different security domains, a concept which can be used for defining trust levels or the availability of encryption mechanisms / keys. The cog wheels indicate processing tasks. The Processing Flow Optimization is responsible for deciding on which processing location a processing task gets deployed.

For this purpose, the Processing Flow Optimization gets an abstract task topology that describes what information sources and processing units are needed and how the information has to flow between them and finally to the sink of the receiving application. This abstract task topology has to be mapped to a concrete topology such that each processing unit is mapped to a processing location in order to be deployed there. This mapping is to be done according to an optimization criterion expressed as an optimization function, but taking into account additional constraints, in particular security aspects. Figure 6 shows an example of an abstract task topology mapped to a concrete topology. Note that some of the nodes on the right side are not trusted and could therefore not be selected for the mapping.

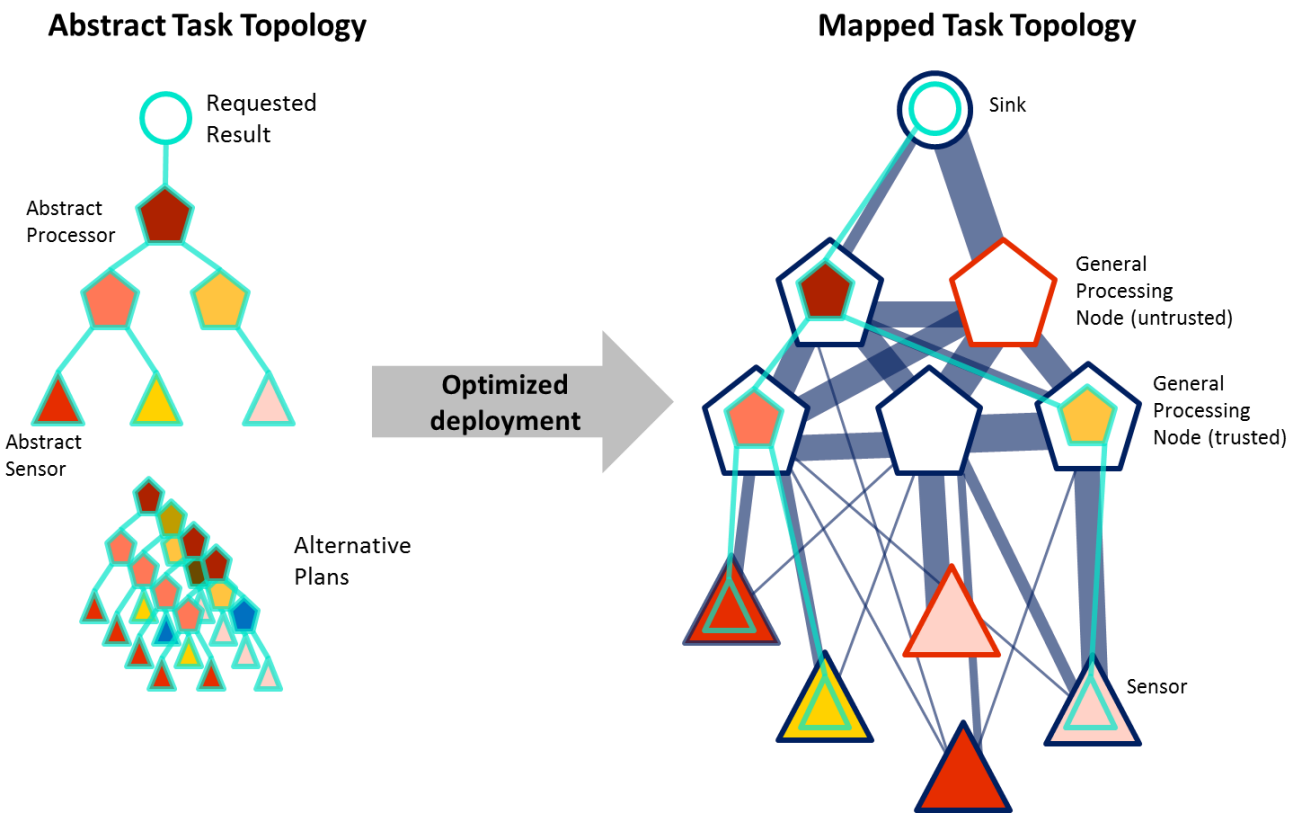

Figure 6 Example of mapping abstract task topology to concrete topology

4.2 Example Use Cases

In the following, we sketch two smart city use cases that have been taken from D2.1 [1], Section 2.2. Based on these we give examples of abstract task topologies and discuss constraints as well as possible optimization criteria. The general setting of the use cases is a Smart City Information Centre with a dashboard application giving an overview of the situation in the city with the possibility of zooming into a geographic area for a selected topic.

4.2.1 Aggregated Energy Consumption

The first use case is about visualizing energy production and consumption. This could be initially visualized on a per district granularity with the possibility of zooming into sections or even city blocks. The original information may have been collected on a per building, per apartment or even per room basis for a Smart Energy Management use case, which is by far too fine-granular and has related privacy issues. Therefore, only the aggregated information could be made available to the Smart City Information Centre. This requires

some processing within the trusted part of the Energy Management subsystem and a suitable access control making sure that only the aggregated information is made available to the Smart City Information Centre.

Figure 7 sketches an example, how the information could be aggregated on different levels. The dashed-pointed line indicates that the processing up to this level has to remain in the security domain *Domain B*.

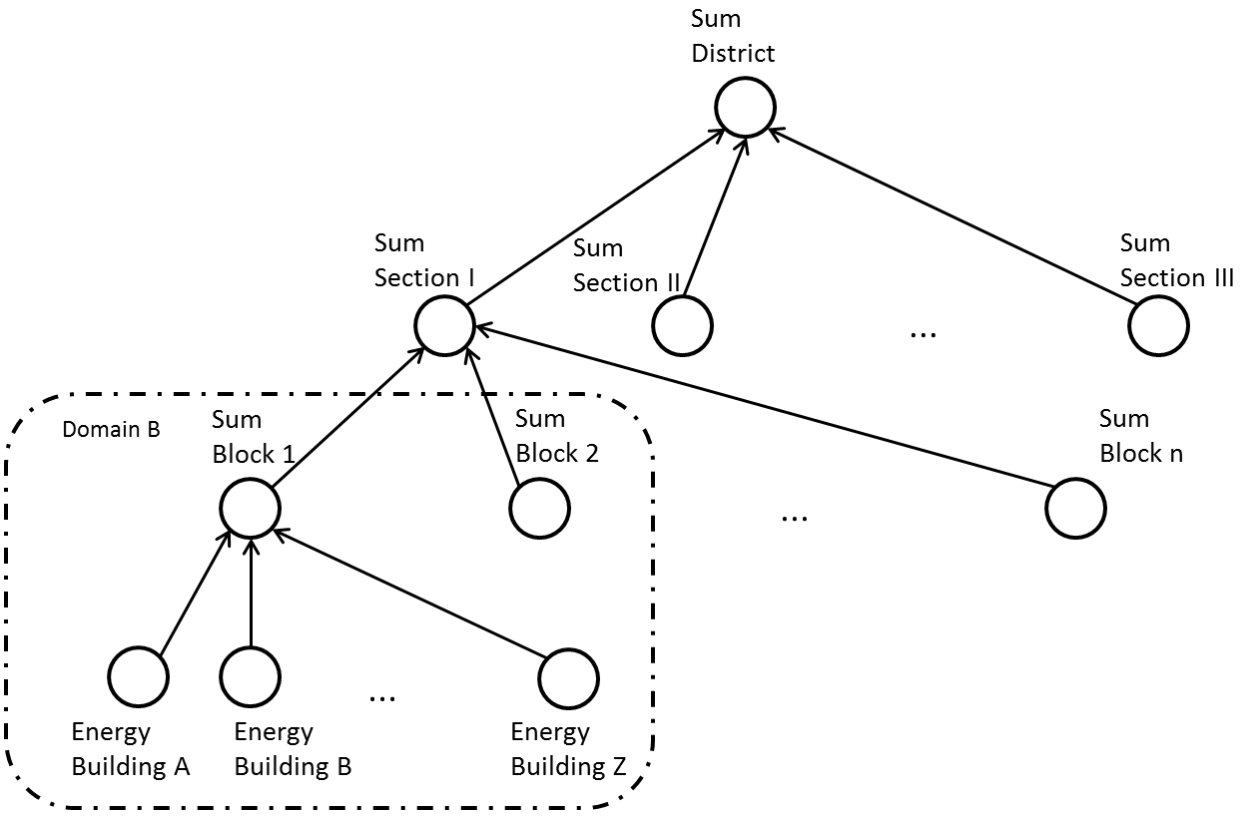

Figure 7 Aggregated Energy Consumption Use Case

Apart from the primary energy stakeholders like energy providers and energy consumers, this information could be relevant for supervisors, policy makers, planners, as well as citizens in general. Especially in the case when the energy consumption has to be restricted as it was the case in Japan after the Fukushima nuclear disaster such information can provide the necessary transparency.

4.2.2 Traffic Situation in the City

The second use case is about visualizing the traffic situation in the city. It could be first visualized on a district level, requiring the processing of the more detailed information. If a higher granularity is required, the information could be provided on a per street section per direction basis. However, this should be done in an anonymized fashion. Depending on how the information is collected, the individual vehicle may be visible within the traffic subsystem, which should not be provided to the Smart City Information Centre. So again, processing within a trusted subsystem and access control is required.

Figure 8 sketches an example, how the traffic information on different levels in the city can be aggregated and assessed. Again, some information needs to be processed within security domain *Domain G*.

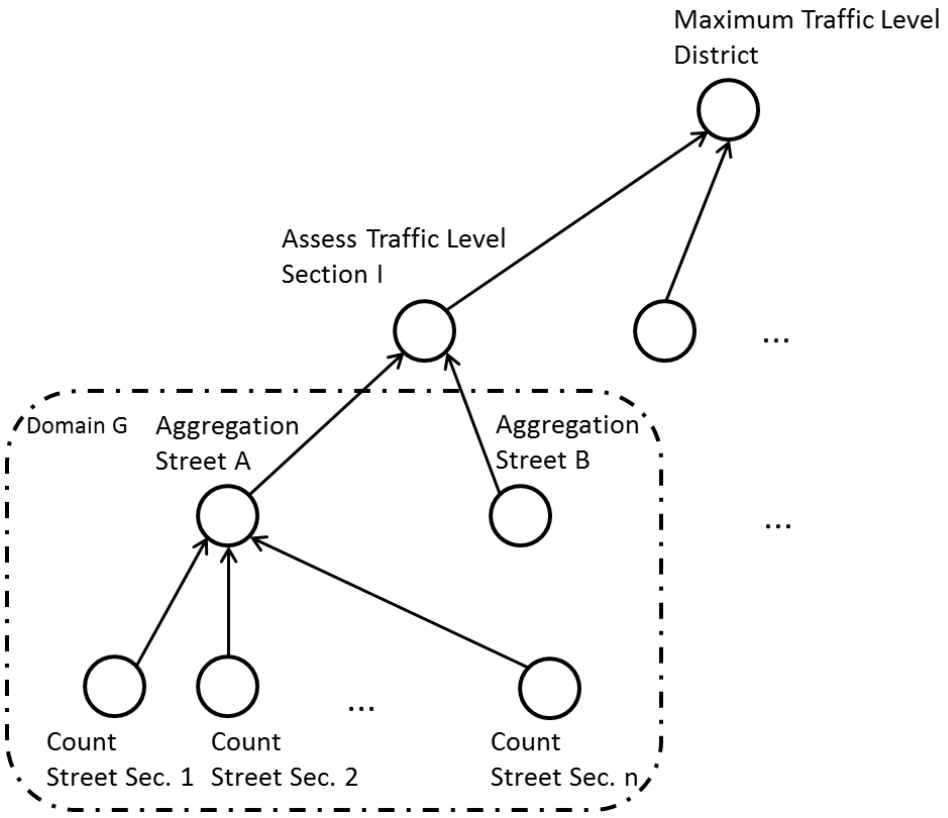

Figure 8 Traffic Situation Use Case

The traffic information is relevant for a number of stakeholders from the traffic department, to transport providers, citizens and the police.

4.3 Mapping Abstract Task Topology to Compute Node Topology

The abstract task topology can be represented as a directed graph in which the vertices represent the tasks and the edges correspond to the flows between them. Tasks that only have an outgoing flow represent sources, tasks with only incoming flows represent sinks (i.e. correspond to the final recipients of information), whereas tasks with both incoming and outgoing flows are processing tasks.

The abstract task topology has to be mapped to the compute node topology for execution. The compute node topology can also be represented as a directed graph with the vertices representing the compute nodes and the edges representing the communication paths between them. In case the communication relationship between two compute nodes is symmetric, there are edges going in both directions. Compute nodes can have the role of source nodes providing information flows (e.g. sensor nodes), the role of processing nodes (i.e. nodes on which a processing unit can be deployed) and the role of sink nodes, which act as the final recipient of information flows. The same compute node can play multiple roles at the same time.

The task for processing flow optimization is to find a mapping between an abstract task topology and the compute node topology in such a way that it is optimal with respect to an optimization function. Based on the explanation above, the mapping can be modelled as the mapping between two graphs, which will be discussed in detail in the following sections.

4.3.1 Requirements on Mapping Abstract Processing Flow Graph to Compute Nodes

It must be possible to map the abstract processing flow graph on the compute node topology.

It must be possible to specify constraints, especially security constraints, in particular that certain information has to be processed within a security domain.

It must be possible to use different cost functions for optimizing the mapping.

4.3.2 Related Work on Subgraph Matching and Optimization

On first sight, it seems that we have an instance of a subgraph isomorphism problem, i.e. the question is whether the graph, representing the abstract task topology can be mapped to a subgraph in the compute node topology graph, so that the tasks can be deployed on the compute nodes. Figure 9 shows an example instance of a subgraph isomorphism problem for undirected graphs.

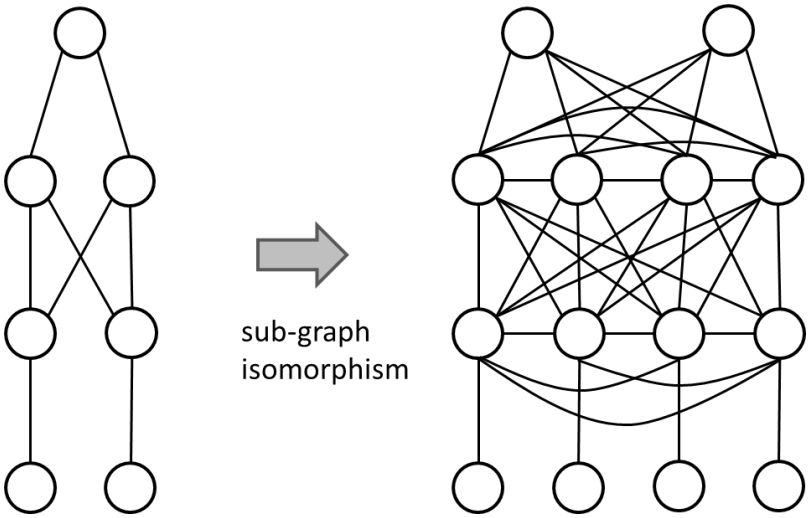

Figure 9 Subgraph isomorphism problem

The subgraph isomorphism decision problem, i.e. "does a sub-graph exist that is isomorphic to a given graph", is NP complete. This can easily be seen as the well-known clique problem, which is NP-complete but can be reduced to the sub-graph isomorphism problem in polynomial time. To check whether a clique with k vertices is included in a graph, a complete graph with k vertices could be constructed, which would then be checked for sub-graph isomorphism [4].

The subgraph isomorphism decision problem is also NP-complete for general directed graphs. A non-directed graph can be transformed into a directed graph by replacing all edges with the two directed edges in both directions. Given we have a solution for the directed graph, we would also have one for the non-directed case and thus the directed subgraph isomorphism problem is also NP-complete.

When taking a more detailed look, it becomes clear that the different vertices in the abstract task graph could be mapped to the same vertex in the compute node graph as a compute node can execute multiple tasks. This requires adding self-looping edges to each vertex of the compute node graph as shown in Figure 10.

According to Shen & Tsai [5], the resulting problem is a weak homomorphism as defined below.

Definition ([5]): "Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs where V_i (i=1, 2) are the vertex sets and E_i (i=1,2) are the edge sets of G_1 and G_2, respectively. G_1 is weakly homomorphic to G_2 if there exists a mapping (possibly many-to-one) M: $V_1 \rightarrow V_2$ such that if edge (a, b) $\in E_1$, then edge (M(a), M(b)) $\in E_2$, and we say that there is weak homomorphism from G_1 to G_2."

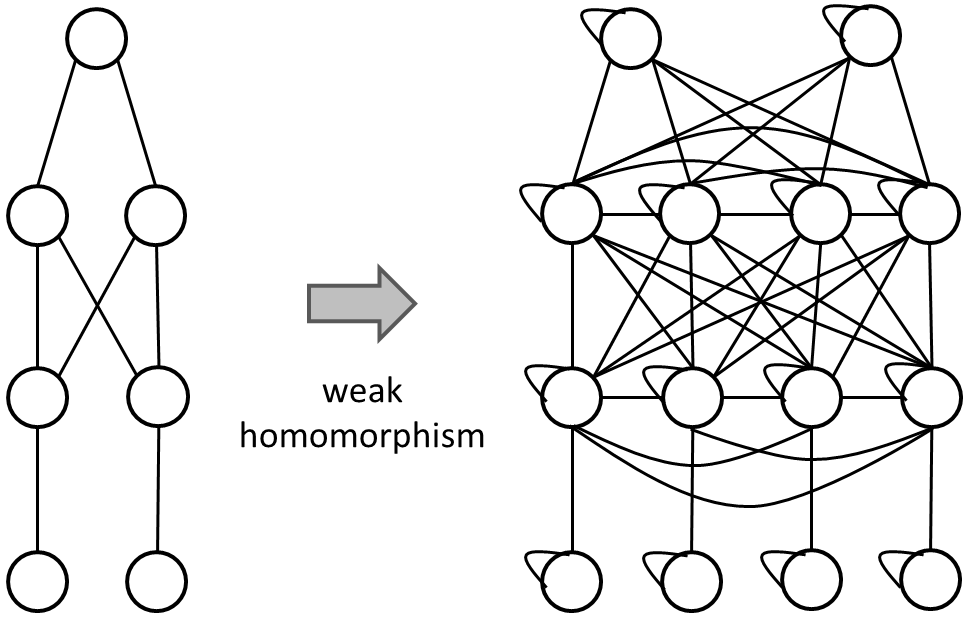

Figure 10 Weak (sub-)graph homomorphism

While this significantly changes the complexity of the decision problem (if you can map all tasks to the same node, assuming sufficient resources are available), the optimization problem remains equally hard.

Shen & Tsai [5] solve a task assignment problem, where a task consisting of different dependent modules is assigned to a heterogeneous set of processors. The processors are connected through non-identical communication links and the modules communicate with each other using the communication links. The goal is to optimize the overall execution time, i.e. minimize the maximum overall execution time of any processor including communication.

The approach proposed for the optimization of this task assignment problem is to use the A* algorithm, which is a state-search approach guaranteed to find an optimal solution using a cost function based on the respective compute and communication times. The search step by step maps nodes, expanding always in the overall "most promising" direction. This is accomplished by using an evaluation f(n) = g(n) + h(n), where g(n) represents the current mapping costs of the partial mapping, whereas h(n) is a lower bound of the costs to complete the mapping. Setting h(n) to 0 results in the uniform cost search, which is guaranteed to find the optimum, but inefficient. The better the h(n) estimates the lower bound of the costs, the more efficient is the A* algorithm.

Our mapping problem significantly differs from the described problems, i.e. we have continuous processing of information flows instead of finite tasks. In order to apply it, a specific cost function needs to be found, for which there is a good h(n) function, estimating the lower bound estimate for the remaining mapping. Since we would like to explore different cost functions for optimizations, it is not straightforward to apply the approach in our case and thus we explore other options.

Taura & Chen [6] propose a heuristic algorithm for mapping communicating tasks on heterogeneous resources. Their target is grid applications, i.e. heterogeneous computing nodes connected through communication links with different properties.

As shown in Figure 11, task graphs and resource graphs are modelled as weighted graphs. In the resource graph, the weight assigned to a vertex represents the compute capacity of a processor (i.e. the amount of computation that can be performed in a unit of time) and the weight assigned to an edge represents the links communication capacity (the amount of data that can go through the link in a unit of time). In the task graph, the weight assigned to a vertex represents the task's compute requirement (the amount of computation that must be done so that the task makes a unit progress) and the weight assigned to an edge represents the task's communication requirement (the amount of data that the task has to communicate to make a unit progress).

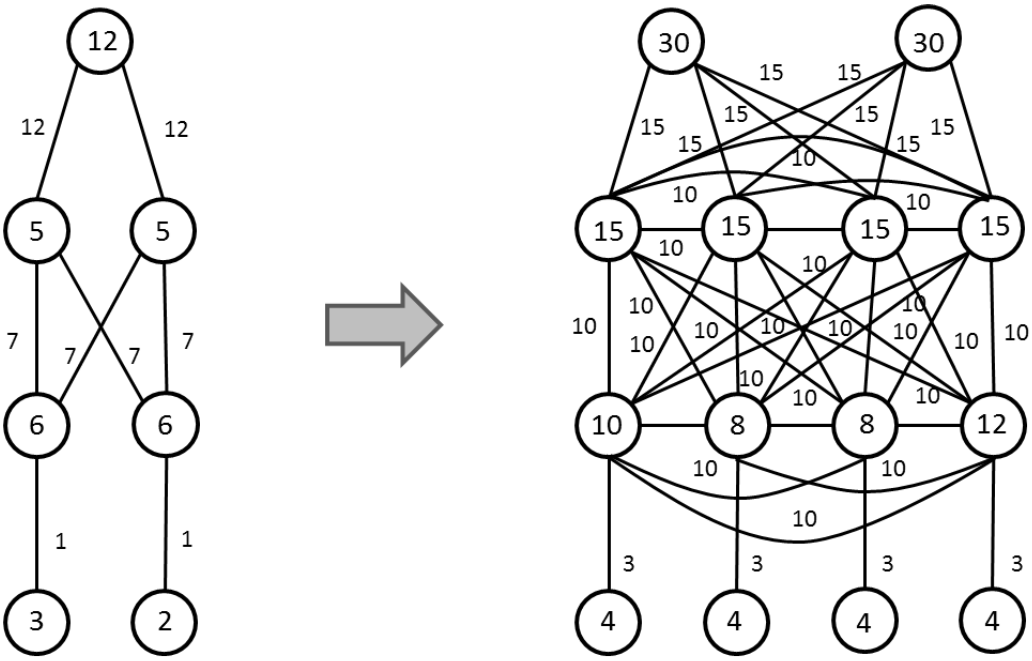

Figure 11 Task graphs and resource graphs modelled as weighted graphs

The approach especially targets communication-intensive tasks, where mapping intensively communicating tasks to different nodes easily becomes a problem. The underlying idea is thus to find highly-connected clusters of tasks and map these to the same processor or a set of well-connected processors.

In order to find clusters, stochastic flow injection [6] is used, that iterates through the following steps:

1) Randomly pick two vertices s and t of the given graph G
2) Find the shortest path between s and t
3) Decrement the weights of all the edges on the path by a small constant Δ
4) Remove edges whose weight become zero or negative
5) Repeat 1-4 until the graph becomes disconnected

The underlying idea is that only a small number of edges will connect clusters (as illustrated in Figure 12). As they are used often on a shortest path their weight will get decremented and eventually such an edge will be removed.

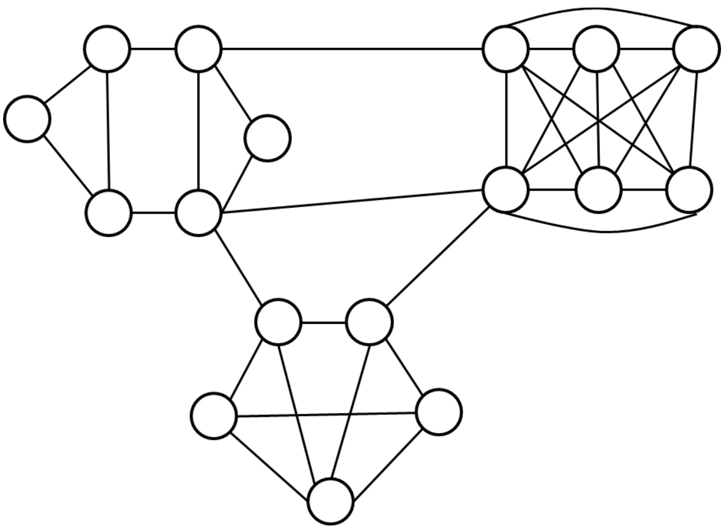

Figure 12 Clusters connected by small number of edges

The clustering provides the order in which the tasks are mapped. At each step of the mapping, the following values are taken into account: the occupancy resulting from the current partial mapping, a hypothetic

occupancy estimating a prefect mapping while ignoring communication and an estimate of a hypothetic occupancy induced by communication. For each step the maximum of the values is recorded and finally the mapping option that gives the minimum of these maxima is selected. Finally, an improvement step attempts to improve the solution by identifying and removing bottlenecks.

The proposed approach seems to be suitable for cases, where tasks can be relatively freely mapped like in a grid or a cloud, but it is not immediately clear how to easily include further restrictions like the location of the sources of information and constraints like security domains. Also, the optimization function is fixed on taking into account processing and communication capacity, making it difficult to integrate other aspects, e.g. energy consumption.

4.3.3 Approach to Mapping Abstract Task Topology to Compute Node Topology

The situation we have is that parts of the abstract task topology mapping are fixed due to sources like sensor nodes being fixed to a network as well as a geographical location. The requester fixes the sink where the flow terminates. In addition, constraints pertaining to security domains limit the choices significantly. Furthermore, the processing nodes (compute nodes on which processing units can be deployed) are very heterogeneous ranging from severely constrained sensor nodes, to gateways, individual servers and cloud nodes. An example topology is shown in Figure 13. The abstract task graph topology is modelled as a directed acyclic graph, whereas the compute node topology is modelled as a general directed graph. The first is due to the fact that processing flows flow in a direction from the sources via the processor to the sink, whereas in the second case, some connections may be unidirectional, e.g. from the sources to the gateway, whereas other connections, e.g. in the cloud go in both directions.

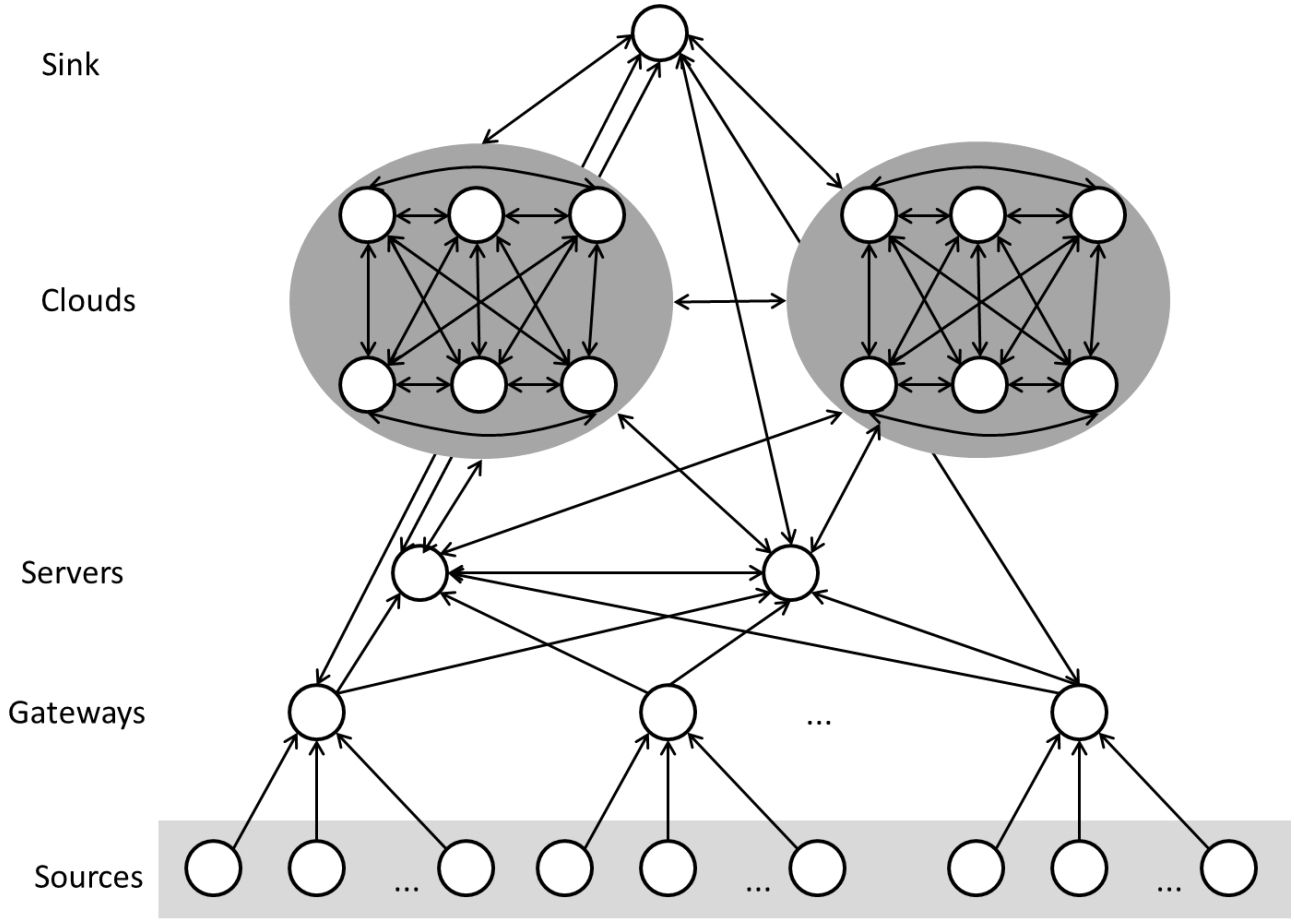

Figure 13 Example topology

Due to the overall heterogeneity on one side and the relative homogeneity and good connectivity of (cloud) nodes, we will evaluate a two-step approach for the mapping.

In the first step we collapse nodes with the same characteristics and good connectivity, e.g. cloud nodes or a server cluster, into single "power nodes". Then mappings are created taking into account all constraints, using the optimization function to select the optimal mapping. The optimization function can take into

account different aspects of the nodes as well as the connections between the nodes. Important aspects are processing power and energy consumption as well as delay and bandwidth. Also different levels of dynamicity regarding the aspects may be considered, i.e. the generally available bandwidth vs. the currently available bandwidth taking into account existing load.

The order in which tasks are mapped is important as in certain cases the search space can be pruned. Compute nodes form a kind of hierarchy, ranging from sensor nodes to gateways, special servers and the cloud. Following the abstract task topology modelled as a directed acyclic graph from the leaves representing sources (i.e. having only outgoing flows), if one task has already been mapped to a gateway, for any node consuming the flow it produces it will not be mapped to the sensor node anymore. Other relations have to be further investigated, but at least it is unlikely that a flow from the cloud will go back towards the gateway – at least unless the sink is located there.

In the second step, all the tasks mapped to the "power nodes" in the first step are mapped to individual nodes, possibly using an approach described in the previous section. The algorithm is shown in pseudo code below.

```
/* G_Task = <V_T, E_T>: abstract task topology graph, annotated with constraints
   G_Nodes = <V_N, E_N>: compute node topology graph, annotated with properties,
                         Including security domains
   opt(m)- cost function used for optimization */

create_mapping(G_Task,G_Nodes, opt): mapping
{
      G_Reduced = collapse(G_Nodes);
      mapping = {};
      mapping = create_high_level_mapping(G_Task, G_Reduced, opt, mapping);
      mapping = create_fine_grained_mapping(G_Nodes, G_Reduced, opt, mapping);
      return mapping;
}
```

Overall, it has to be assessed whether this two-step approach can handle typical smart city IoT scenarios or whether more specific approaches as described in the previous section are needed, which would then likely be fixed on one specific optimization function that is intrinsic to the mapping.

Figure 14 shows an example mapping for the aggregated energy consumption use case presented in Section 4.2. On the left side, (parts of) the abstract task topology specified by the user is illustrated, which is mapped to the compute node topology (partly) illustrated on the right side.

The abstract task topology specifies sources providing the energy consumptions of buildings. The energy of buildings A to Z is to be summed up to provide the energy consumption of Block 1, the same for Block 2 etc. These are further to be summed up for sections and districts. The individual information regarding the Buildings has to stay within security domain DomainB, i.e. it is required that the processing has to take place within this domain.

The compute node topology shows the nodes on the different levels. The source nodes are shown at the bottom and the different types of processing nodes are shown above. The communication links are directional – most links are bi-directional, i.e. have two arrows. Only the links from the sources to gateways are unidirectional. For visibility reasons only a single communication link is shown from outside nodes to the cloud and vice-versa.

In the first step of the mapping, the nodes of the respective clouds are collapsed into one power node each, i.e. the decision will only be whether tasks are mapped to this cloud or not. Then the nodes are mapped starting from the bottom of the abstract task topology. The sources have a fixed mapping and due to the specification of the constraint regarding security domain DomainB, the Sum processing unit has to be mapped to the gateway node. For the mapping of the higher level Sum processing units, there are no specific

constraints, so they could be mapped to gateways, servers or cloud nodes. The actual choice depends on the optimization function. In the example both the Section summing processing units are mapped to the cloud as well as the District summing processing units.

In the second step, the processing units that were mapped to the cloud have to be mapped to specific cloud nodes. In the given example, the processing units for summing Section I and the District are both mapped to the same cloud node.

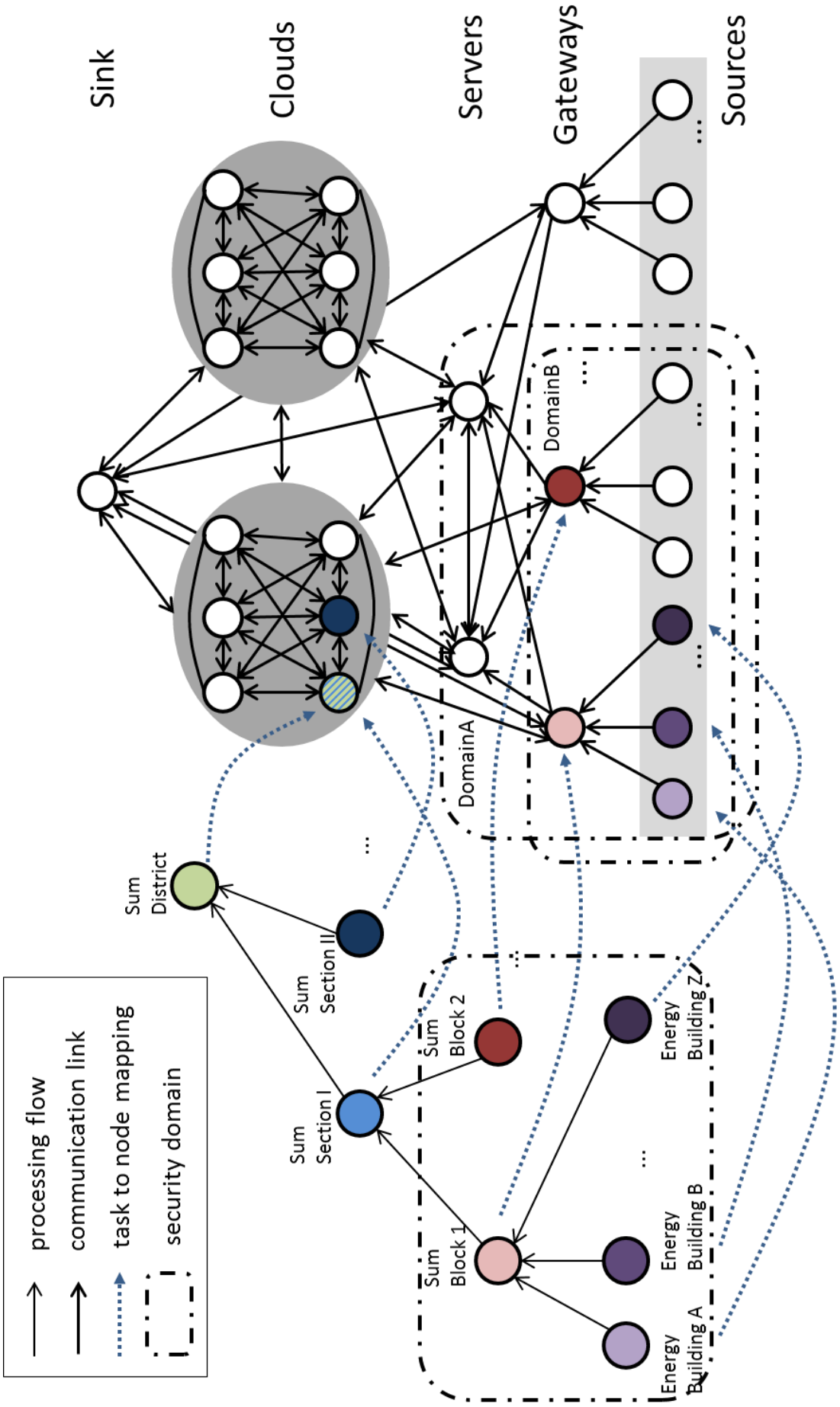

Figure 14 Example Mapping for Aggregated Energy Consumption Use Case

5 Environment for Dynamic Deployment

In order to deploy the processing units on the processing nodes (compute nodes that allow the deployment of processing units) identified by the processing flow optimization, an environment for the dynamic deployment of components is needed. In this section different choices for dynamic deployment environments are considered. Note that the focus of this work is on the processing flow optimization, so an existing solution for dynamically deploying processing components on compute nodes will be chosen that is suitable for demonstrating the overall solutions, but this solution may not be suitable for all types of relevant scenarios, e.g. highly constrained devices like sensor nodes.

5.1 State of the Art

There are different approaches for dynamically deploying code on compute nodes. These range from complete virtual machines to containers, dynamic component systems and specific solutions. Virtual machines are relatively heavy-weight as they run a complete operating system, but they provide complete flexibility as the operating system, configuration, programming language can be freely chosen. Containers are more light-weight, but they are bound to a specific operating system, e.g. Linux. Different containers share the same operating system kernel and libraries [7], the programming language can still be freely chosen. Dynamic component systems, e.g. OSGi, are based on a specific virtual machine and thus typically a given programming language. However, they can run on any operating system on which a virtual machine is available. Specific (device) management systems may allow the dynamic deployment of code, typically based on specific hardware. For constrained hardware like sensor nodes, only the last option may be available, as all other approaches are too heavy-weight. In the following we look at some typical examples in more detail.

5.1.1 OSGi

The OSGi Alliances has specified the OSGi Service platform [8] implementing a dynamic component system based on Java programming language. Applications or components are provided in the form of bundles for deployment. Components are exposed as services, hiding their internal implementation. Bundles can be remotely installed, stopped, updated and uninstalled without reboot. The advantage of OSGi is its platform independence, i.e. it runs on all systems for which a Java Virtual Machine is available. A disadvantage is that only the Java programming language is supported.

5.1.2 Docker

Docker is the currently most popular container approach. Applications run in a light-weight container on the Linux operating system. Docker uses the isolation features offered by the Linux kernel to isolate one container from other containers or applications running directly on the host Linux operating system [9]. Using Docker, containers can be started within a timeframe of seconds.

5.1.3 VMWare

VMWare is an example of a full virtual machine approach. A VMWare image contains a full operating system which then runs on a completely virtualized hardware. The result is that a VMWare image is highly portable and can run as a virtual machine on any computer that can provide the virtualized hardware. Using VMWare, the user has complete control, from the operating system, its configuration to the programming language used for writing the application. The price is the relatively heavy-weight nature of such a virtual machine that typically takes minutes for start-up.

5.2 Discussion

For demonstrating the dynamic deployment, we will use a container approach. A full virtual machine approach seems far too heavy-weight for the purpose, whereas the container approach is flexible and offers

fast deployment. OSGi would also be an option, but limits the programming language to Java. With any of these approaches, no resource-constrained devices can be supported, but as already stated above, the focus of this work is not on the dynamic deployment part, but rather on the processing flow optimization, so for a proof-of-concept, it should be sufficient.

6 Framework Integrating the Components that Implement the System and Management Mechanisms

In the previous sections, components implementing different system and management mechanisms for large scale IoT systems have been presented. In order to make use of these, they have to be integrated into a common framework, making them accessible to applications in a uniform way.

6.1 Detailed Architecture

Figure 15 details the high-level architecture presented in Figure 1. The main difference is that two concrete SMARTIE components have been selected for integration. The first component is the RD that provides the Discovery functionality within SMARTIE and thus will also be used here for looking up and discovering suitable processing flow sources (e.g. sensor nodes) and processing nodes respectively.

The second component needed for authentication and authorization is the Capability Manager, which provides the user application with the token for accessing the Processing Flow Optimization.

In Section 6.2 the required functional interactions are described (shown in blue in Figure 15), in Section 6.3 the interactions needed for securing Processing Flow Optimization are presented (shown in orange in Figure 15).

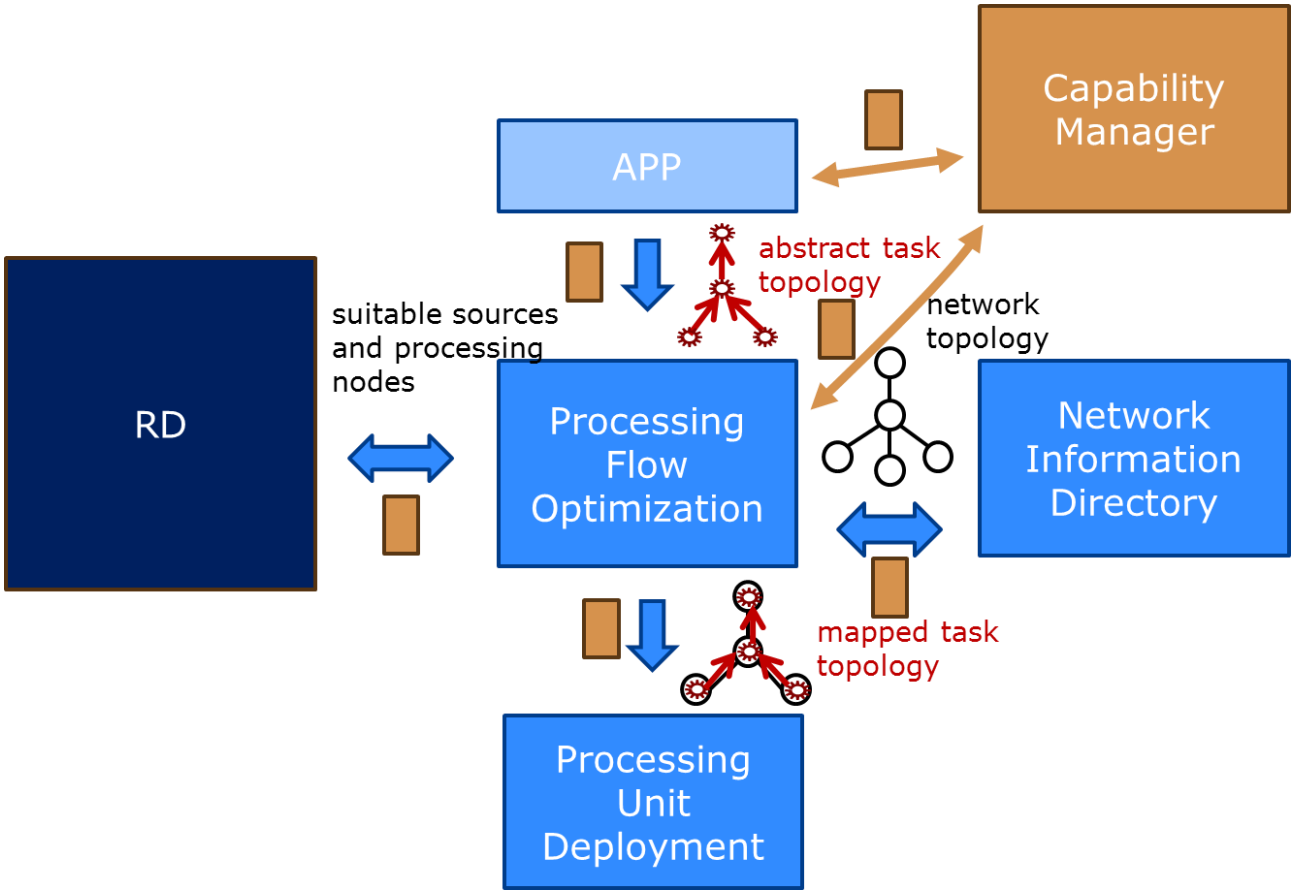

Figure 15 High-level architecture with SMARTIE components

6.2 General Interaction Diagram

In the following we list the high level steps that are needed for Processing Flow Optimization (PFO) and the setting up of processing flows:

1) Application calls PFO with an abstract task topology, including constraints to be deployed
2) PFO extracts information regarding required sources from the abstract task topology and looks them up in RD
3) PFO discovers candidate processing nodes for deploying processing units as specified in the abstract task topology from RD
4) PFO retrieves the network information within the scope of the sources and candidate processing nodes from the Network Information Directory (NID)
5) PFO creates mapping of abstract topology to the compute node topology based on the optimization function
6) PFO calls Processing Unit Deployment (PUD) with the mapped task topology
7) PUD requests processing units (code) from Processing Unit Storage
8) PUD deploys processing units on processing nodes according to the mapped task topology
9) PUD configures the processing units and sources, initializing the processing flows

Figure 16 shows the detailed interactions for optimizing and setting up the processing flows, including the important parameters and return values.

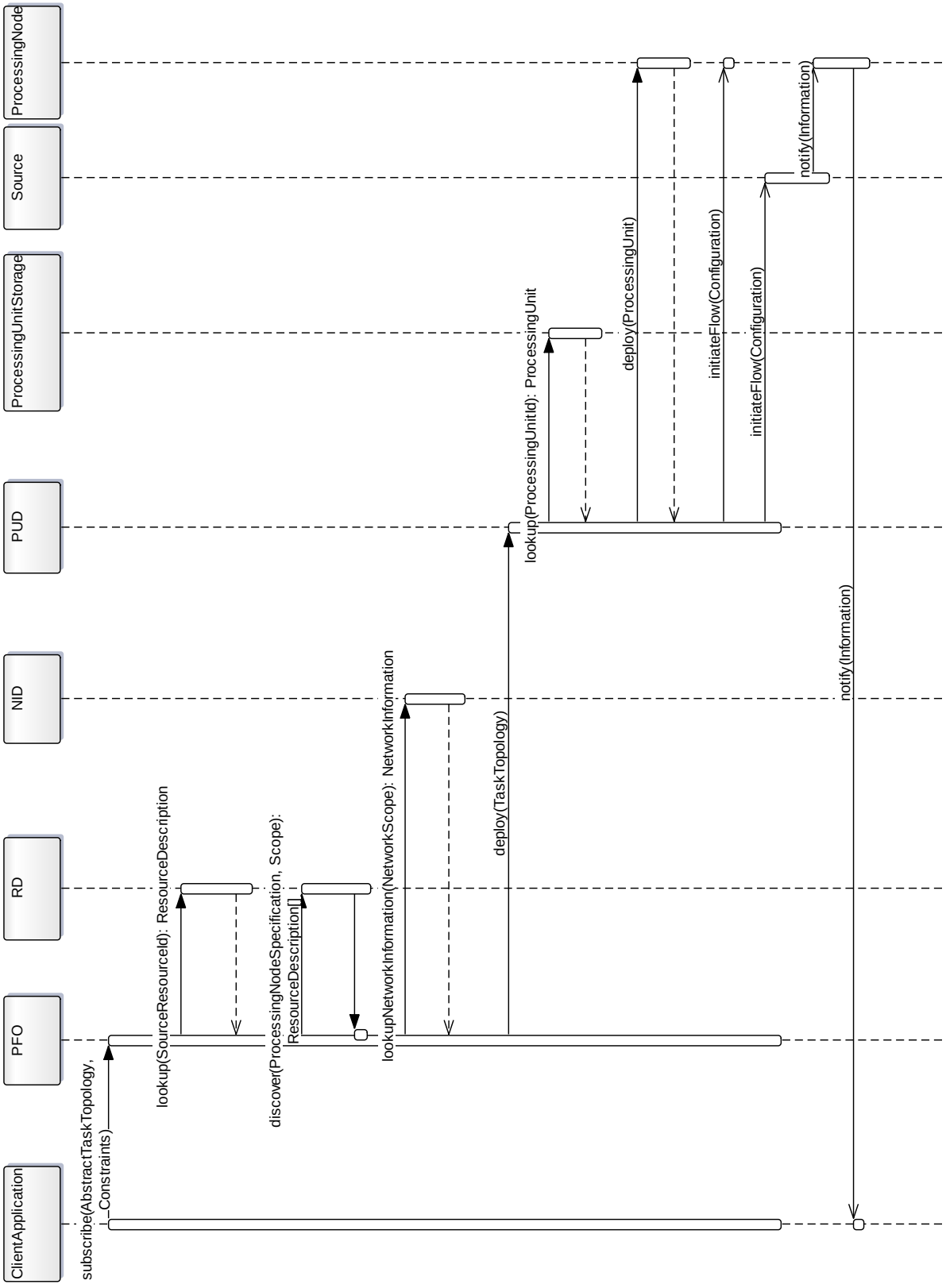

Figure 16 Detailed interactions for optimizing and setting up processing flows

6.3 Securing Processing Flow Optimization

In Section 6.2, only the interactions needed from a functional point of view have been shown. To secure the overall process in accordance with the overall SMARTIE approach some additional steps are necessary that ensure the proper authentication and authorization of all requests triggered by the user. A key component in this respect is the Capability Manager that provides the tokens necessary for authorizing the respective requestors to make a specific request.

The PFO has a special role in this process as it orchestrates most of the requests to other components, acting on behalf of the user application. In the following, we consider the PFO to be a service inside the SMARTIE platform. As a trusted platform component, it can act as a kind of administrator when requesting tokens, but still acting on behalf of the user, e.g. providing a userId in the requests to the RD to filter the results. A simplification of the process could be achieved if, for example, all compute nodes use the same resource name, as then one "master token" could be enough to read from all of them, instead of needing N tokens with some special identifier in the token.

In the following, the high-level steps from Section 6.2 are enhanced with the steps that are needed for securing processing flow optimization. These steps are shown in italics.

0) User Application requests token from Capability Manager that authorizes it to request processing flow optimization from PFO.

1) User Application calls PFO with an abstract task topology, including constraints to be deployed *and token from Capability Manager*

   a) PFO checks the token to determine whether User Application is authorized for request

2) PFO extracts information regarding required sources from the abstract task topology and looks them up in RD *providing token and userId*

   a) RD checks the token to determine whether the User Application is authorized for this request

   b) RD filters according to the access rights of the user

   c) PFO gets authorization information for these sources (tokens)

3) PFO discovers candidate processing nodes for deploying processing units as specified in the abstract task topology from the RD, *providing token and userID*

   a) RD checks the token to determine whether the User Application is authorized for this request

   b) RD filters according to the access rights of the user

   c) PFO gets authorization information for the processing nodes (possibly a master token is sufficient as mentioned above)

4) PFO retrieves the network information within the scope of the sources and candidate processing nodes from the Network Information Directory (NID) , *providing token and userID*

   a) NID checks the token to determine whether the User Application is authorized for this request

5) PFO creates mapping of the abstract topology to the compute node topology based on the optimization function

6) PFO calls the Processing Unit Deployment (PUD) with the mapped task topology, *providing all required tokens (PUD, processing nodes and sources) and userID*

   a) PUD checks the token to determine whether the User Application is authorized for this request

   b) PUD request Token(s) for accessing Processing Unit Storage

7) PUD requests processing units (code) from the Processing Unit Storage, *providing tokens*

   a) *Processing Unit Storage checks the token to determine whether the User Application is authorized for this request*

8) PUD deploys processing units on processing nodes according to the mapped task topology, *providing respective tokens*

a) Processing nodes check tokens

9) PUD configures the processing units and sources, initializing the processing flows, *providing the required tokens*

a) Processing units and sources check the respective tokens before initialization

Figure 17 shows the detailed interactions for optimizing and setting up the processing flows, including the security aspects, especially the interactions with the Capability Manager.

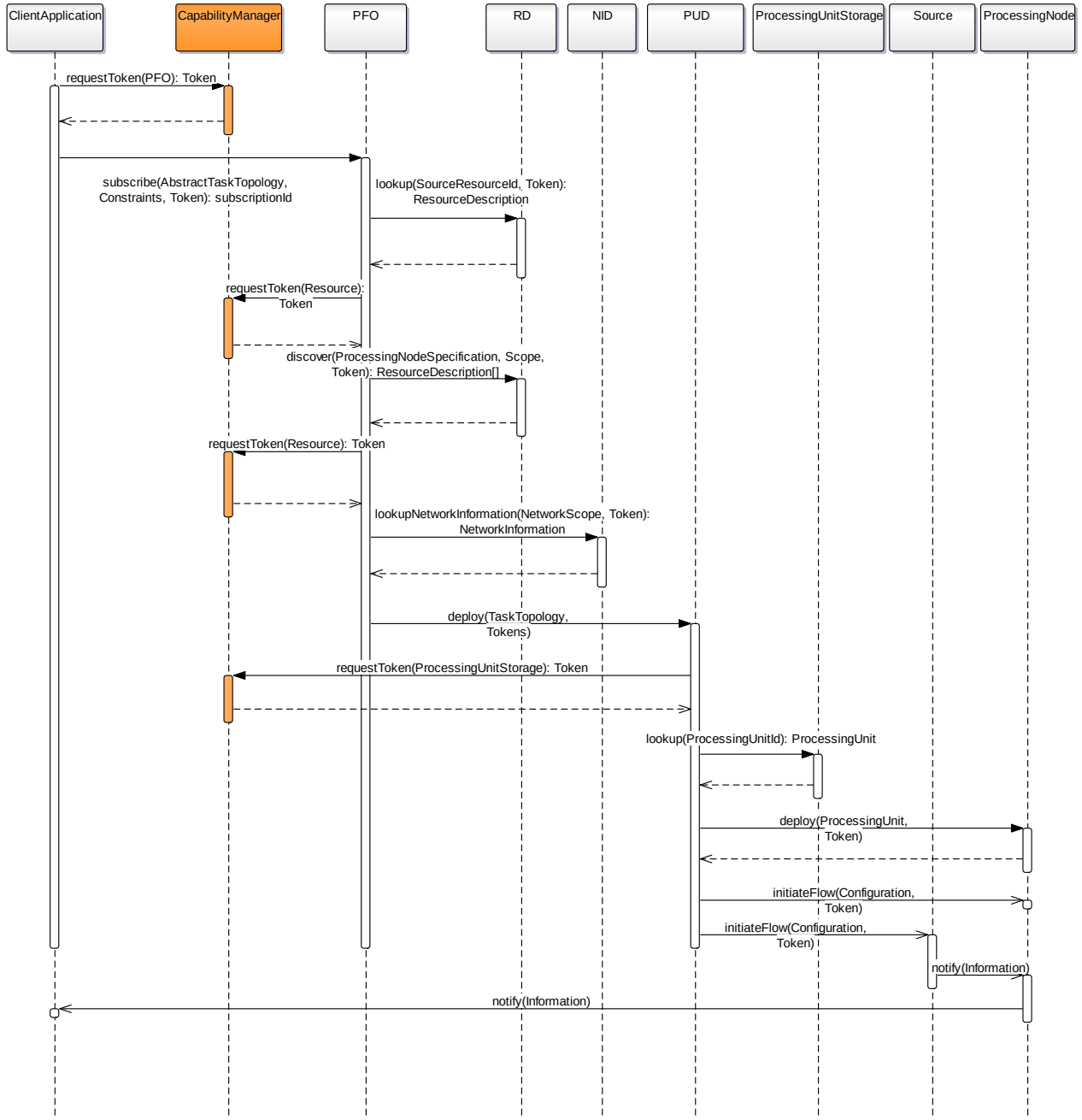

Figure 17 Detailed interactions with security-related aspects

7 Conclusions

This deliverable describes the design of the service management for discovery and retrieval that primarily targets the case of large scale IoT deployments for the Smart City. The key components selected for this purpose are the RD for resource discovery and look-up, the network information directory, the processing flow optimization and the processing unit deployment. For securing the access to the components and the information they provide, the components interact with the Capability Manager.

The main focus of the work presented here is on the processing flow optimization, i.e. the planning where to place required processing functionality optimizing according to an optimization function and taking into account constraints, especially security constraints like security domains that can be used for modelling trust. We propose to tackle the hard optimization problem by using a two-step mapping approach. Since the types of nodes to which the processing tasks can be mapped are rather heterogeneous, e.g. sensor nodes, gateways, edge servers or cloud nodes, we first combine similar nodes into "power nodes" and do the mapping on this basis and only in a second step we map the processing tasks assigned to the power nodes to actual nodes. Finally, taking into account the constraints and starting the mapping from the sources should enable an efficient solution for practical problem instances. This will be evaluated as part of WP5.

References

[1] SMARTIE Deliverable D2.1, Use Cases, 2014

[2] SMARTIE Deliverable D2.3, Initial Architecture Specification, 2015

[3] SMARTIE Deliverable D4.2, Event Processing and Framework Specification for Privacy and Access Control, 2015

[4] Subgraph Isomorphism Problem, Wikipedia, https://en.wikipedia.org/wiki/Subgraph_isomorphism_problem

[5] Chien-Chung Shen; Wen-Hsiang Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," in *Computers, IEEE Transactions on* , vol.C-34, no.3, pp.197-203, March 1985

[6] Taura, K.; Chien, A., "A heuristic algorithm for mapping communicating tasks on heterogeneous resources," in *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th* , vol., no., pp.102-115, 2000

[7] Charles Babcock, "Containers Explained: 9 Essentials you need to know", http://www.informationweek.com/strategic-cio/it-strategy/containers-explained-9-essentials-you-need-to-know/a/d-id/1318961

[8] OSGi Specifications, http://www.osgi.org/Specifications/HomePage

[9] Kyoung-Taek Seo, Hyun-Seo Hwang, Il-Young Moon, Oh-Young Kwon, Byeong-Jun Kim, "Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud", Advanced Science and Technology Letters, Vol.66 (Networking and Communication 2014), pp.105-111

[10] Hernández-Ramos, J.L.; Bernabe, J.B.; Moreno, M.V.; Skarmeta, A.F. Preserving Smart Objects Privacy through Anonymous and Accountable Access Control for a M2M-Enabled Internet of Things. Sensors 2015, 15, 15611-15639

[11] Constrained RESTful Environments (CoRE) Link Format, https://tools.ietf.org/html/rfc6690

[12] Hypertext Transfer Protocol -- HTTP/1.1, https://tools.ietf.org/html/rfc2616

[13] Java SE Security, http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html

[14] Open Mobile Alliance, "Open Mobile Alliance," 05 2012. [Online]. Available: http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20101207-C/OMA-TS-NGSI_Context_Management-V1_0-20100803-C.pdf.

