Service Discovery and Composition in Smart Cities

Nizar Ben-Sassi[1], Xuan-Thuy Dang[1], Johannes Fähndrich[1], Orhan-Can Görür[2], Christian Kuster[1], and Fikret Sivrikaya[1]

[1] GT-ARC gGmbH, Berlin, Germany
[2] DAI-Labor, TU Berlin, Germany

Abstract. The ongoing digitalization trend has given rise to the concept of smart cities, targeting the interconnection of city infrastructure and services over a digital layer for innovative technological solutions as well as improvements on existing facilities in cities. This article investigates the critical information system constituents of smart cities that facilitate the holistic integration of its ecosystem and resources with the aim to foster dynamic and adaptive software. We identify three main enablers in this direction: (i) semantic functional description of city objects, representing physical devices or abstract services, (ii) a distributed service directory that embodies available city services for service lookup and discovery, (iii) planning tools for selecting and chaining basic services to compose new complex services. We provide an overview of the approach adopted in our ongoing smart city project for each of these three dimensions. We also revisit the available tools and results from the research literature on each topic.

Keywords: smart cities, service discovery, service composition, semantic service description, adaptive planning

1 Introduction

The perennial trend of urbanization has been transforming human life for many decades, whereby the city infrastructure and services become increasingly more integral parts of our lives in the form of transportation systems, energy systems, and many more. More recently, the rising trend of digitalization brings new dimensions to urbanization; a concept typically captured by the term "smart cities". Advancements in information and communication technologies (ICT), such as the Internet of Things (IoT) [1], cloud computing, and mobile computing, provides a foundation for the digitalization of city systems. This often results in better resource utilization among infrastructure providers through autonomous processes and more flexible interaction between citizens, authorities and other stakeholders through ubiquitous access to information. However, the abundance of city data makes information management one of the central challenges in enabling cross-domain collaboration. With the lack of unified data and service integration platforms, the transformation to digital cities often results in specific applications that represent isolated service and data silos.

In general, the information systems of a smart city can be in the form of physical devices with ICT capabilities, entirely virtual online services, or a combination of both, which we commonly refer to as *Smart City Objects (SCO)* in this article. A smart city object can be as simple as a temperature sensor providing data to the cloud or as complex as a trip assistance service that stems from a well-crafted composition of many other SCOs from the transport, traffic, energy, and environment domains. The project Intelligent Framework for Service Discovery and Composition (ISCO), presented in this paper, aims to develop an open and extensible platform together with supporting tools for the creation and holistic interconnection of smart city objects from different sectors and stakeholders. This objective translates into moving away from fragmented IoT solutions and isolated data silos in cities towards a more integrated and harmonized smart city ecosystem. ISCO enables stakeholders from diverse domains to provide novel, efficient and dynamic services, optionally composed of other existing services in the smart city. One of the main challenges of such a platform is ensuring its scalability while enabling efficient development, deployment, and discovery of smart city objects.

In this paper, we review such smart city challenges, solution approaches, and suitable tools as well as an in-depth overview of our proposed contributions in the ISCO Project. In particular, after a high-level overview of the ISCO approach to smart city service integration (Section 2), we present a unified semantic model for SCOs based on semantic web technologies (Section 3), a scalable distributed architecture for SCO discovery based on information centric networking (Section 4), and a two-level planning approach for orchestrating through both generic and application-specific service ontologies that also incorporate quality of service (QoS) attributes (Section 5).

2 Background and ISCO Approach

The smart city concept has been attracting strong attention of the research community from a large variety of research fields, particularly in the computer science and information systems disciplines [2, 3]. On the other hand, a growing number of smart city initiatives is spawned around the world in recent years. A report published by the European Parliament [4] as early as in 2014 explores smart city initiatives in terms of their impact on the objectives defined in the European Growth Strategy 2020, covering more than 50 smart city projects that were conducted in 37 cities. Given the vast breadth and depth of the smart city scope, we do not intend to provide a comprehensive background here, but refer the readers to the surveys on the topic, e.g., [2, 5, 6], for a detailed coverage of the socio-technical systems of smart cities.

The increasing adoption of digitalization and the existing smart city applications have pointed out several challenges in building such IoT frameworks in cities, ranging from security and network reliability to data management aspects. While there have been many approaches proposed to deal with these challenges, a suitable collaboration scheme between the existing IoT solutions, heteroge-

neous devices and services remain as an open issue [6]. There is a growing need in harmonizing and integrating the solutions from different perspectives and domains in order to prevent further fragmentation in this field. ISCO Project's approach towards addressing this need of integrated solutions is to design and develop open platforms and tools for interconnecting heterogeneous entities in a city through what we call *smart city objects*. While the project also covers the networking and security aspects for the interoperability and accessibility of all city objects, our focus in this paper remains on the three core modules of ISCO – represented in Fig. 1 by the Smart City Object API, Service Directory, and Agent Domain components.

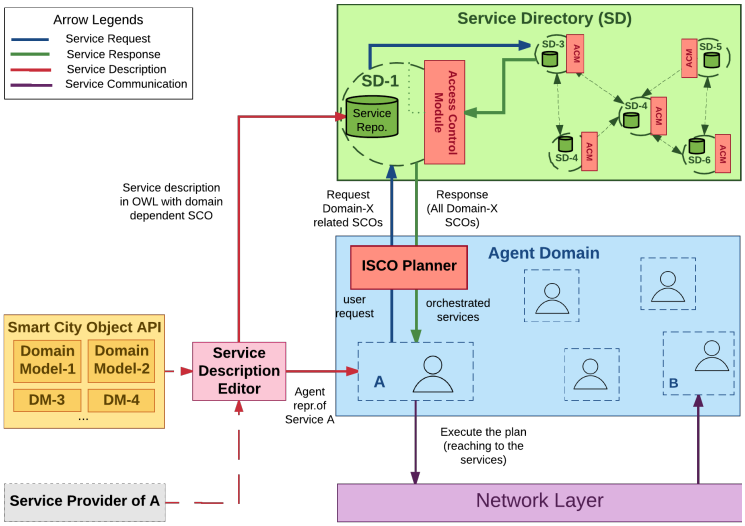


Fig. 1: ISCO Architecture: Overall Interaction of the Main Components

In the high-level service workflow of ISCO presented in Fig. 1, service providers can introduce their service descriptions using our relevant domain models, after which a software agent representation of the service is automatically created under the ISCO agent domain. The description is also saved within the Service Directory (SD) in the Web Ontology Language (OWL) format similar to [7]. The SD is structured as a dynamic collection of distributed nodes (typically hosted by service providers or other market players throughout the city) and serves as the repositories and request points for registered services. For example, *Service A* in Fig. 1 is registered to the service directory node SD-1 as the closest SD instance. Every service agent contains an instance of the ISCO Planner to request and orchestrate services. Once a user makes a request, the agent forwards it to find domain-related SCOs from its registered node. Then, the node may ask other nodes in the SD, using an information-centric routing overlay, as described in Section 4, finally returning a set of matching services. The access control module

filters out services that are not authorized for the requester. Finally, the domain that includes the services with authorized access are returned. ISCO Planner can now process the services first to filter based on their qualities then to find an applicable orchestration of the services based on the user request. We present the details on the inner-workings and the interactions of these components in the remainder of the paper.

3 Semantic Description of Smart City Objects

Our fundamental understanding of a smart city is the interconnection of heterogeneous IT-enabled entities, namely physical devices as well as virtual services, that provide a certain functionality: Smart City Objects (SCO). The functionalities provided by the individual entities overlap and are not known a priori, which complicates and impedes the efficient use of them. Consequently, to enable other entities to utilize such a SCO, an appropriate technical representation is needed. In addition, an API should facilitate the creation of and access to SCO for ISCO-compatible applications; something that other approaches often neglect. The representation and the concept of the API are discussed in the following.

For a reasonable way of sharing knowledge and modeling SCOs in a heterogeneous and dynamic environment, the usage of ontologies is the de-facto approach; domain and data models described in an ontology provide a sophisticated semantic description that can be parsed by an application. Existing approaches that model the *Internet of Things* (IoT) domain in a descriptive way commonly adopt the IoT-Lite ontology [8], which is utilized in FIESTA-IoT[3]. The lightweight approach gives a good insight on how to efficiently classify the different accessible devices in a smart environment and what properties such a model should provide at least, but lacks a functional description that can be called by a requester.

In describing the SCOs in terms of their functionality, it seems the best approach to see them as (web) services. The service-oriented computing community developed well-established standards to describe and invoke functionalities, such as WSDL/SOAP and REST services, but these standards do not suffice a smart city. Semantic descriptions of software services provide the needed additional information layer, for which several approaches have been proposed in recent years. Amongst others, these are OWL-S [9], WSMO [10] and SAWSDL[4], which extend typical service information by preconditions and effects, and define input and output information in a more expressive ontology language, such as the Web Ontology Language (OWL) [11]. However, these semantic approaches only address web services and not the smart city domain in particular. Therefore, in our approach we extend the OWL-S ontology for smart cities. Nambi et al.[12] describe a semantic knowledge base for IoT and use the OWL-S ontology to describe services, similar to our approach. However, the authors focus on describing the domain holistically with contextual information, and therefore create a rather complex multi-layered ontology; furthermore, orchestration of services is neglected in the presented architecture.

[3] http://fiesta-iot.eu/

[4] http://www.w3.org/2002/ws/sawsdl/

Our extension is more geared to IoT-Lite with focus on orchestration; it integrates devices and further extends the concept of a service by attributes we identified as needed by a SCO, e.g., the location or the accessibility. An overview of our ontology can be seen in Fig. 2, where the central class is *SmartCityObject* that is a subclass of the OWL-S classes *Service*, *ServiceProfile*, *Process* and *Result*. This means that in terms of OWL-S, SCOs represent both the service as well as the underlying process with its result and, on the other hand, guarantees that SCOs can be used by systems that understand OWL-S services.

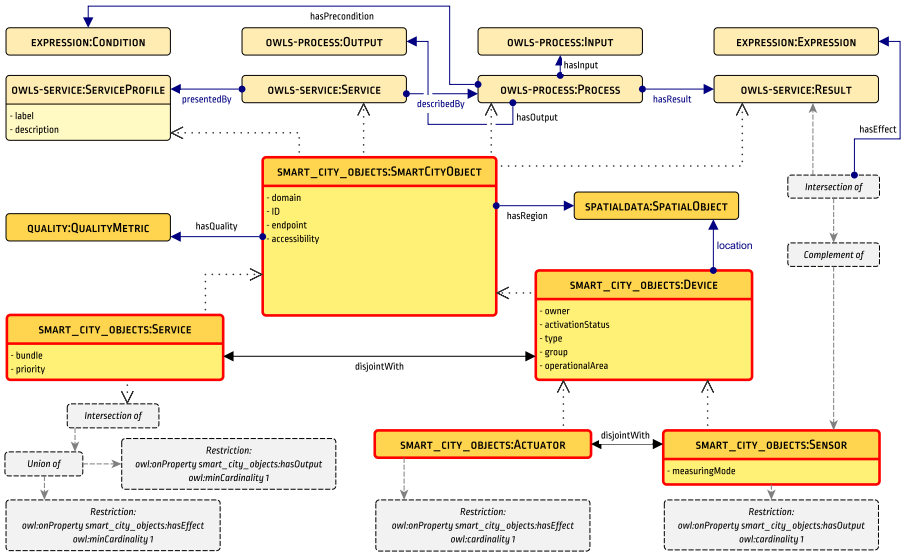


Fig. 2: Centralized Overview of the Smart City Object Ontology

Besides virtual or digital services, a significant part of a smart city is represented by devices. Since we model the domain in terms of their functionality, we distinguish here between *Actuators* and *Sensors* that only differ in terms of what they provide: An actuator changes the world's state, thus has an effect, while a sensor measures a kind of quantity, thus only provides some output. By this definition we are able to describe smart city devices in the same way as virtual services.

The large amount of potential SCOs with similar or identical functionalities in a smart city pose a challenge on service discovery and planning operations. Here the design of the semantic model plays a crucial role, as the amount of returned SCOs can be reduced drastically with the possibility for a fine-grained description of the required service. In this connection, our model allows assigning SCOs to (multiple) sub domains of the universal smart city domain, whereby we refer to the overview made by Neirotti et al. [13] for the individual domains.

Furthermore, devices can refer to a type in a taxonomy to better describe their nature. As another important feature, non-functional attributes that facilitate a better selection of SCOs are related to the geospatial information of the entities, i.e., the regions for which a SCO provides its functionality as well as the location of physical devices, whenever applicable. Lastly, management attributes such as the provider of the SCO, the type of accessibility to provide security, attributes for grouping SCOs and unique identifiers are adopted to ensure a working and efficient autonomous utilization of such entities.

Even for entities that are functionally equivalent, non-functional attributes, or quality of service (QoS) parameters, can be used to further narrow down the required SCO. While there are many existing approaches for QoS ontologies [14, 15], a description of our QoS model would go beyond the scope of this paper. Nevertheless, the processing of these criteria will be addressed in Sec. 5.

Smart City Object API. The Smart City Object API facilitates the usage of SCOs by city applications and stakeholders. Because of the special requirements of a smart city, especially the huge number of entities, the SCOs are stored in a distributed service directory, as will be explained in the next section. The API provides methods for easy access to city objects: SCOs can be queried directly by their ID, or a more generic search can be triggered based on attributes such as the domain, input and output parameters, quality parameters, and overall score, after which the matching SCOs are returned. The API also converts between the ontology representation used in the service directory and an object-oriented representation that can be used directly by application developers. In summary, the SCO API functions as a bridge between the other components in the ISCO architecture to allow for easy implementation of ISCO-compatible applications that build on the functionalities provided by smart city objects.

4 Scalable Service Lookup and Discovery

In this section, we discuss challenges of managing smart city information and propose a design for city-scale distributed directory of smart city objects for lookup and discovery, which facilitates dynamic end-to-end service composition.

Smart city systems have to deal with billions of devices and services, which are often combined together in order to provide city applications. Such devices and services may need to be uniquely identifiable for tracking or establishing connections, for which the *naming* and *addressing* mechanisms are designed. The former is concerned with creating a labeling scheme or attribute that differentiates an object in a local or global scope, while the latter is concerned mainly with locating the object or service and enabling their interactions. Most of the existing IoT or smart city solutions are built on the currently well established architectures and data communication technologies such as Service Oriented Architectures (SOA) and the Internet Protocol (IP). While those technologies have well-defined mechanisms to address and locate devices and services, SCOs

pose new challenges to their management, ascribed to the smart city service requirements: i) heterogeneity of devices and services, ii) complex human-service interactions and behaviors, iii) cross-domain integration, and iv) highly dynamic and mobile environment.

Due to heterogeneity and cross domain requirements, smart city IoT solutions rely on an object discovery infrastructure to provide descriptions about their attributes, location, and access methods, among others. Depending on the application, discovery can be a stand-alone service or integrated with the entity management and gateway functions of an IoT middleware. Nevertheless, it aims at providing scalable services for object registration, mapping, and lookup. As such, object descriptions can be distributed based on logical domain, geographical location, or platform specific hierarchy:

- *Domain-specific discovery* builds on the Domain Name System (DNS) of the Internet and is adopted by some projects such as IoT6 by leveraging IPv6 and proposing service discovery (DNS-SD)[5] and mDNS[6] discovery protocols. The Object Name Service (ONS) [16] used in SmartAgriFood is a similar service to DNS for discovery and resolve physical object with EPC code. A local ONS server looks up product descriptions for scanned code by mapping it to a set of resource descriptions (URI) provided by external services.
- *Geolocation based* discovery is common in device centric and location based applications. The objects are addressed based on their notation of geographic points, areas, or network cluster. While the indexing and geo-discovery are straightforward, additional resolution infrastructure is still required to provide operational details of the resources, as in, e.g., IoT-A and BUTLER projects.
- *Service Directory(SD)* A directory structure holding rich descriptions of IoT entities is often required in addition to previous distribution approaches. Beside accessibility description, attributes about the entities and their relationships provide data needed for, among others, management, service composition logic of the applications. Semantic web approach is adopted by the projects and is referred as semantic discovery. OWL-based ontologies capture models of physical, logical entities and their relationships.

ISCO Service Directory based on an ICN Overlay We propose an IoT service directory (SD) in ISCO that self-organizes the distributed storage and retrieval of smart object descriptions. Its flat architecture makes the directory eligible for universal service discovery for IoT by removing the dependency on discovery mechanism from specific applications and domains. Before we present our SD design in ISCO, we first provide a short introduction to ICN as an important enabler for our design.

Information-Centric Networking: The underlying principle of ICN is that a communication network should allow a content consumer to focus on the data it needs, named content, rather than having to reference a specific, physical

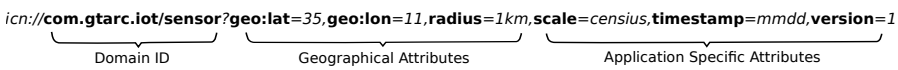
[5] http://www.dns-sd.org/

[6] http://www.multicastdns.org/

location where that data is to be retrieved from, i.e., named hosts, as in current Internet architecture. ICN offers a wide range of benefits, e.g, content caching to reduce congestion and improve delivery speed, simpler configuration of network devices, and building security into the network at the data level. Communication in ICN is driven by data consumers, through the exchange of Interest (INT) and Data (DATA) packages. Both types of packets carry a name that identifies a piece of data. The consumer sends an INT with the name of the data it needs. When an intermediate node receives the INT, it looks for the data in its content store (CS). If the data is not found, it forwards the INT to the next nodes and keeps track of the incoming and outgoing network interfaces for this data in pending interest table (PIT). A series of such forwarding actions creates a breadcrumb path the INT has passed. When the INT arrives at the source node, the requested data is put into DATA and sent back the path towards the consumer. Intermediary nodes on the path cache the DATA in their CSs for subsequent INTs.

We apply ICN's data centric paradigm, more specifically the *Named Data Networking*[7] as a realization of the ICN approach, for the distribution of object descriptions among SD-Nodes in our service directory design. The solution can take advantage of the aforementioned ICN features for IoT requirements due to the data-centric nature of many smart city and IoT applications. Based on those features, refined mechanisms are designed for SD functionalities, i.e., developing attribute-based object query methods and content caching strategies for reduced storage overhead as well as increased responsiveness and accuracy.

ICN based Naming Scheme for City Objects. In the service directory infrastructure, the data to be exchanged are descriptions of services and devices, which mainly contain various attributes. Using ICN enables the SD to decouple the data from locations of the nodes that store the data while taking advantage of ICN forwarding and caching mechanisms. As detailed previously, each SD-Node is an ICN router, which serves the requests for object's description by its name, or forwards the requests towards other nodes holding the description. Therefore, the design of a naming scheme affects the performance of objects discovery. ICN naming adopts the semantics of Universal Resource Identifier (URI) scheme. However, the host part does not imply location of the resource, but rather identifies its owner or search domain. The *attributes* part enables the expression of resource attributes that can be used to look up and discover the resources regardless of where they are stored. An ICN name is shown below, which contains rich semantics describing a sensors domain, location, type, etc.



Matching of query attributes and the semantic descriptions is handled by a matcher component in each SD-Node. For this purpose, attribute names can refer to their semantic description by using popular ontology prefixes, e.g., the *geo* namespace in the example. Depending on the use-case, a strategy to store

[7] https://named-data.net/

descriptions and to forward the requests based on object attributes can be dynamically configured. Additionally, various caching and forwarding strategies can be designed to best serve the query demands and SD infrastructure performance.

Service Directory Node Architecture. The ISCO service directory is constituted by a distributed collection of SD nodes as depicted in Fig. 3. The design of an individual SD-Node, which contains semantic descriptions of SCOs, is shown in Fig. 4. We employ a triple store for the storage and query of SCO attributes. Various interfaces are implemented for respective transport protocols for the query and distribution of the descriptions. Each functionality is implemented as a modular component based on the OSGI platform architecture. Descriptions of most important components are provided next.

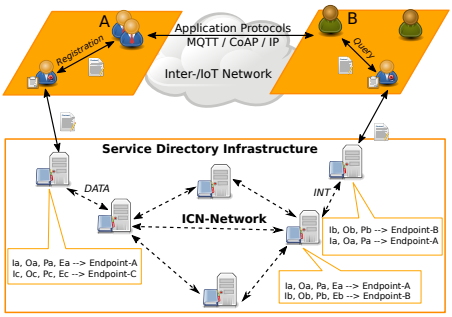


Fig. 3: ICN-based Service Directory

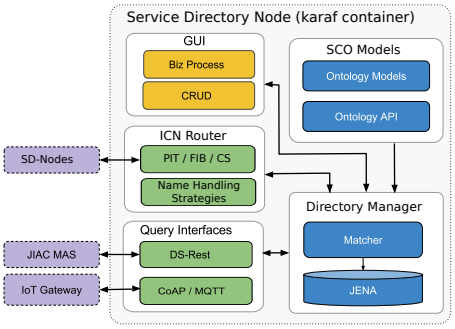


Fig. 4: SD-Node Architecture

Triple Store (TDB) is a component of the Jena [8] project, which serves as a high performance RDF store for the directory server. It provides an API as well as a SPARQL interface for storage and query of semantic descriptions.

Matcher component implements mapping methods between requested search attributes and suitable SCO descriptions, which are potential search results. It handles queries from IoT service components and other SD-Nodes received through different query interfaces.

ICN Router enables connectivity between SD-Nodes with ICN transport protocol to form a distributed SD infrastructure. Discovery of SCO descriptions is realized by the exchange of interest messages with SCO names. The available CCN-Lite [17] solution is extended with the implementation of strategies for forwarding requests and caching SCO descriptions among SD-Nodes.

Query Interfaces provide distributed application protocols, which allow higher level services to access SCO descriptions and ontologies. The protocols are implemented for various application transport protocols, e.g., CoAP, MQTT, Rest. To enable semantic expressions and functional description in the queries, the REST-

[8] https://jena.apache.org/documentation/tdb/index.html

desc [18] approach is applied. It enables certain matching rules to be embedded, which improves the accuracy of SCO matching and discovery.

Scalability. An ICN architecture, in contrast to a host-centric one, does not dictate a predefined hierarchy, e.g., conformance with IP routing or a specific discovery protocol (DNS), among others. This results in a flat network with self-organized topologies. The attribute-based discovery only depends on how an approach describes the devices and services, specifically, their semantic models, matching approach, and strategies for information organization. Fig. 3 illustrates a distributed SD infrastructure utilized by the ISCO platform, which is based on multi-agent system architecture. The agents on the ISCO platform are logical representations of SCOs. Once a service or device is made available, the agent (A) registers its service in the SD by sending SCO descriptions to a nearby SD-Node making it the source of the SCO description. Due to the flatness ICN transport network, no constraint is given on the placement of the SD-Nodes. Globally dedicated nodes or, if required, managed nodes for each local domain or organization could be used. If a service (B) wants to look for another service's (e.g., A) description, it sends the request to a local SD-Node in an ICN interest message. The request is forwarded to the source SD-Node of the description, which results in replications of the description in the individual caches of SD-Nodes along the query path. Forwarding and caching strategies can be adapted; e.g., the choice of lifetime of the replicas implies a trade-off between the dissemination of descriptions closer to requesting agents, and timeliness, consistency of the information. Some caching approaches are discussed in [19]. Moreover, the self-organizing topology allows additional SD-nodes to be added or removed as required by deployment scenarios, querying patterns, among others. Applying cloud computing or container technologies (e.g., Docker) enables elastic provisioning of SD services.

5 Service Composition and Planning

Web service composition (WSC) has been widely applied [20] to create new value-added services from existing atomic ones. The QoSs of all services involved in this composition affect the quality of the composite service. In this section we present the *ISCO middleware planning layer* which applies WSC to IoT components. Planning in IoT environments is a challenging task: IoT instances may appear, disappear or move, thus, the promised Service Level Agreement (SLA) can change over time. Applying WSC in such a fragile heterogeneous and dynamic environment requires an adaptive and self-optimizing architecture. Therefore, we adopt an extended version of the MAPE-K [21] architecture by IBM, which enhances the system with significant adaptation capabilities.

Implementing new services for smart cities is a complex task, as service developers have to identify and bind the required set of services and sensors with the highest QoS/QoD out of millions of possible SCOs at design time. The implemented services have a static binding to dynamic SCO instances and have there-

fore low adaptation capabilities and stability. Maintaining such services is also costly. The service developers have to monitor the system dependencies to detect qualitative or functional changes. Those changes require a code revision for the sake of replacing misbehaving or outdated SCOs. The ISCO middleware eases the implementation and deployment of such complex services by abstracting the service and device layer allowing service developers to focus on the functional and qualitative definition of their services.

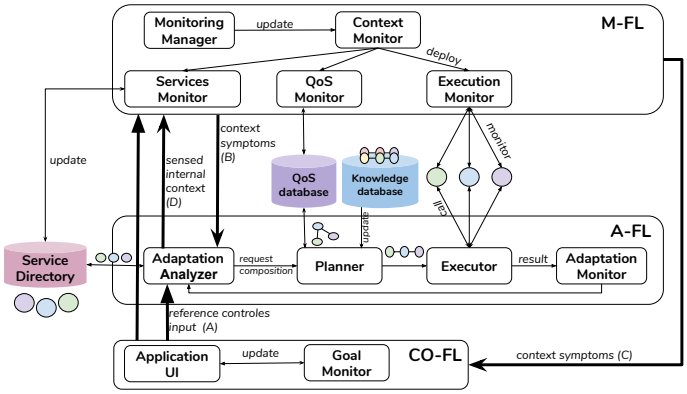


Fig. 5: An overview of the ISCO planning layer components

The ISCO middleware planning layer is responsible for generating a service composition, involving different SCOs, that satisfies the specified – functional and qualitative – application requirements at runtime. The adaptability of this module is crucial, as it has to suit different contexts and serves a wide range of applications with varying goals. We are therefore applying the DYNAMICO [22] design guidelines for adaptive systems to enhance the planning layer adaptiveness. This reference model defines three levels of dynamics: the *Objective Feedback Loop (CO-FL)*, which is responsible for managing changes in control objectives, the *Adaption Feedback Loop (A-FL)*, which models the adaptation mechanisms, and the *Monitoring Feedback Loop (M-FL)*, which tracks context and objective changes.

Objective Feedback Loop. In dynamic software systems, the adaptation goals (or objectives) should be defined and monitored. These control objectives can define functional system requirements or refer to non-functional system properties (e.g., QoS) [23]. The monitoring of these adaptation goals needs an explicit formalization, which e.g., is accomplished in AI planning through a goal state. The goal state contains the facts the system should achieve. In ISCO, we are using an IOPE (input, output, precondition and effect) representation to define the functional goals (e.g., plan a trip) and QoS to state the non-functional

system requirements (e.g., trip cost and duration). During the execution of software systems in dynamic environments, the adaptation goals might be affected by several changes, including, but not limited to:

- *Goal change:* Because of the changes in the environment, a goal might no longer be valid. Users can also explicitly modify the adaption goals.
- *Goal no longer reachable:* During the execution of a plan, the environment might change, e.g., through disappearing services or changes in the condition of the environment. Those changes might lead to situations where the goal is no longer reachable, given the current plan.
- *Goal order change:* While executing a plan to reach a certain goal, the priority of goals might change, e.g., if the QoS of used services changes. With a change in QoS the goal might become less attractive, and with that, a new goal could be perused. By changing priorities, a certain plan might lose effectiveness or efficiency.

An adapting system, especially in dynamic environments like the IoT, should be able to monitor its goals and evaluate when to start switching objectives. Switching a goal might cause the abortion, adaption or recreation of a plan, e.g., if weather change influences a part of a journey, and riding a bike is no longer a valid option. In the DYNAMICO Framework [22] the CO-FL addresses such problems.

Adaptation Feedback Loop. (A-FL) receives the adaptation goals from the CO-FL and the monitored context information from the M-FL and selects the appropriate adaptation mechanism to maintain or reach the system goals. Our A-FL layer implements different approaches that enable the system to adapt to system-wide changes. In this loop, the adaptation process might be initiated due to changing control objectives or context information. The A-FL has four main components introduced below:

Planner is responsible for combining SCOs by connecting their IOPEs for the sake of generating new composite services that satisfy the client application requirements. Our approach is extending the traditional QoS-aware WSC to support IoT devices and sensors. The generated plans should fulfill the functional and qualitative system requirements defined by the CO-LP. The current implementation is graph based, and uses a Fast Forward planner [24] to reduce the search space and fasten the system response.

Interpreter is responsible for the execution of the composite services. The execution is monitored whereas the current state is forwarded to the analyzer module. This process may fail or the results may deviate from the specified goals. In this case, the adaptation analyzer should trace those deviations and replace the missing or misbehaving components to maintain the system robustness. The analyzer might also stop the execution process if needed, e.g., if the adaptation goals were updated.

Adaptation Analyzer is the central component of the A-LP. This module evaluates the current adaptation goals, selects the most suitable adaptation

mechanism and initiates the adaptation process. It also identifies and deploys the required monitoring modules. This module stores – for each new request – the generated service composition along with its global QoS in the knowledge database. If the system goals or context are updated, the analyzer will first search the knowledge database for a service composition that meets the current system requirements. The freshness of the solution, if found, is controlled. If the solution does not exist or is outdated, the planning process is initiated. The generated plan is then forwarded to the interpreter; finally, if the execution is successful it is stored in the Knowledge database. If the execution fails, the planning process is re-triggered in order to replace the unavailable service(s).

Adaptation Monitor checks the state of the adaption mechanism. The adaption needs to change if the adaption mechanism itself is no longer adequate for the system. This monitoring is done to observe the performance of the adaption mechanisms in case the adaption mechanism itself needs to be adapted. This inadequacy could be the case, e.g., if the goals or the context change faster than the adaption can react. In this case, the adaption mechanism might neglect an optimal solution to speed up the adaption. This adaption mechanism is modeled in the planning component. Depending on the heuristic used, the planning can adapt to goal changes, e.g., if different quality parameters become important, or by adapting the planning parameters like relaxing the optimality of A^* to a ϵ-admissible heuristic [25] to speed up the search for a solution.

Monitoring Feedback Loop. Self-adaptive systems need to maintain their context-awareness relevance, in order to adapt at runtime to changing context. For the sake of preserving the system context-awareness, different monitoring strategies might be applied depending on the current adaptation goals. The M-FL is the context manager in the DYNAMICO reference model (see Fig. 5). The M-FL deploys different context gathers, which monitors the current system context, and reports updates to the A-FL. Our ISCO platform implements four different monitoring components each of which is targeting a specific system component or process:

QoS Monitor is responsible for monitoring the QoS parameters of the supported services and devices. The measured QoS at runtime may deviate from the defined SLA used to generate a service composition, and should therefore be updated. To guarantee optimality, the system has to adapt to these changes. By changing QoS, the service composition has to be adjusted and the monitored QoS has to be taken into account during the planning process. The current version of the QoS monitor supports network-specific global parameters, relying on OpenStack to periodically monitor the throughput, latency, package loss, error rate and availability of the SCO.

Services Monitor – As mentioned before, service developers are able to create new services or update the functional requirements of their services. These

updates have to be considered during the planning phase in order to guarantee the optimality of the final composition. This component observes the service directory and notifies changes to the adaptation analyzer.

Execution Monitor – During the execution of a composite service several issues may arise (e.g., timeout exception, network exception, the returned values does not have the right format). This module reports the tracked issues to the adaptation analyzer, which then initiates the most appropriate recovery process, e.g., replacing unavailable services with similar ones or generating a new sub compositions.

Context Monitor captures changes in the context of the adaption system. This monitoring is done to be able to adapt to changing conditions in the environment, e.g., changing legal rules of the planning domain.

6 Summary and Future Work

In this paper, we present the concept of our ISCO framework, a holistic approach for service discovery and composition in smart cities. The current effort focuses on providing an ecosystem that eases the implementation and deployment of dynamic and self-adaptive software. Our ongoing work tackles the common challenges IoT projects face, which are mainly the lack of integrity and interoperability of cross-domain platforms. The unified SCO, the service directory and the service composition and planning layers are the main components of this middleware. We are currently working towards the development of the different components, and once completed, the adaption capabilities and scalability of the system will be tested in a heterogeneous dynamic testbed with both physical and simulated devices.

References

1. Bibri, S.E., Krogstie, J.: Ict of the new wave of computing for sustainable urban forms: Their big data and context-aware augmented typologies and design concepts. Sustainable Cities and Society **32**(Supplement C) (2017) 449 – 474
2. Gharaibeh, A., Salahuddin, M.A., Hussini, S.J., Khreishah, A., Khalil, I., Guizani, M., Al-Fuqaha, A.: Smart cities: A survey on data management, security, and enabling technologies. IEEE Communications Surveys Tutorials **19**(4) (Fourthquarter 2017) 2456–2501
3. Brandt, T., Donnellan, B., Ketter, W., Watson, R.T.: Information systems and smarter cities: Towards an integrative framework and a research agenda for the discipline. In: AIS Pre-ICIS Workshop-ISCA 2016. (2016)
4. Manville, C., Cochrane, G., Cave, J., Millard, J., Pederson, J.K., Thaarup, R.K., Liebe, A., Wissner, M., Massink, R., Kotterink, B.: Mapping smart cities in the eu. (2014)
5. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future Generation Computer Systems **29**(7) (2013) 1645 – 1660

6. Arasteh, H., Hosseinnezhad, V., Loia, V., Tommasetti, A., Troisi, O., Shafie-khah, M., Siano, P.: Iot-based smart cities: A survey. In: 2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC). (June 2016) 1–6
7. Opdahl, A.L., Henderson-Sellers, B.: Grounding the oml metamodel in ontology. Journal of Systems and Software **57**(2) (2001) 119 – 143
8. Bermudez-Edo, M., Elsaleh, T., Barnaghi, P., Taylor, K.: Iot-lite: a lightweight semantic model for the internet of things. In: IEEE International conference on Ubiquitous Intelligence and computing, IEEE (2016) 90–97
9. Burstein, M., Hobbs, J., Lassila, O., Mcdermott, D., Mcilraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. Website (November 2004)
10. Domingue, J., Roman, D., Stollberg, M.: Web service modeling ontology (wsmo)-an ontology for semantic web services (2005)
11. McGuinness, D., van Harmelen, F.: OWL Web Ontology Language Overview. Technical report, W3C (2004) http://www.w3.org/TR/owl-features/.
12. Nambi, S.A.U., Sarkar, C., Prasad, R.V., Rahim, A.: A unified semantic knowledge base for iot. In: Internet of Things (WF-IoT), 2014 IEEE World Forum on, IEEE (2014) 575–580
13. Neirotti, P., De Marco, A., Cagliano, A.C., Mangano, G., Scorrano, F.: Current trends in smart city initiatives: Some stylised facts. Cities **38** (2014) 25–36
14. Tran, V.X., Tsuji, H.: A survey and analysis on semantics in qos for web services. In: Advanced Information Networking and Applications, 2009. AINA'09. International Conference on, IEEE (2009) 379–385
15. Kritikos, K., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benrernou, S., Brandic, I., Kertész, A., Parkin, M., Carro, M.: A survey on service quality description. ACM Computing Surveys (CSUR) **46**(1) (2013) 1
16. GS1: GS1 Object Name Service (ONS) Version 2.0.1. Ratified Standard **2** (2013)
17. : Ccn lite: Lightweight implementation of the content centric networking protocol. http://ccn-lite.net Accessed: 2017-11-30.
18. Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., Walle, R.V.d., Gabarró Vallés, J.: Capturing the functionality of web services with functional descriptions. Multimedia Tools and Applications **64**(2) (May 2013) 365–387
19. Abdullahi, I., Arif, S., Hassan, S.: Survey on caching approaches in information centric networking. Journal of Network and Computer Applications **56**(Supplement C) (2015) 48 – 59
20. Sheng, Q.Z., Qiao, X., Vasilakos, A.V., Szabo, C., Bourne, S., Xu, X.: Web services composition: A decades overview. Information Sciences **280**(Supplement C) (2014) 218 – 238
21. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (January 2003) 41–50
22. Villegas, N., Tamura, G., Müller, H., Duchien, L., Casallas, R.: DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems. In: Software Engineering for Self-Adaptive Systems 2. Volume 7475 of LNCS. Springer (August 2012) 265–293
23. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Transactions on software engineering **30**(5) (2004) 311–327
24. Hoffmann, J.: Ff: The fast-forward planning system. **22** (09 2001) 57–62
25. Pearl, J.: Heuristics. Intelligent search strategies for computer problem solving. The Addison-Wesley Series in Artificial Intelligence (1985)