3
Related Work

In this chapter, we present several different approaches for distributed reasoning in AmI environments and discuss the advantages and disadvantages of these systems.

3.1
Distributed Reasoning

Middleware systems that give support to ubiquitous computing environments traditionally adopt a centralized approach for their reasoning mechanisms [28], in which a central entity is responsible for collecting the available context data from all sensors and ambient software entities operating in the same environment, and for all the required reasoning tasks, which may include transforming the imported context data in a common format, deducing higher-level context information form the raw context data, and taking context-oriented decisions for the behavior of the system [92]. This is the case of CoBrA [19], CHIL [22] and Semantic Space [32], for example.

However, in AmI environments, applications, services, rules and context information may be partially or fully distributed among the different elements involved. Thus in some circumstances a centralized approach may be inefficient and even infeasible — e.g. if not all context information is available at the node in charge of reasoning. In such environments, distributed reasoning is necessary to address the complexity that arises from the coexistence of different elements that collect, store, process, exchange and reason about context data [33]. Hence, there are some approaches for distributed reasoning that try to overcome this limitation, such as Gaia [18], OWL-SF [34], DRAGO [93], P2P-DR [28] and P2PIS [94]. In the following sections we discuss the main features of each of these aforementioned solutions.

3.2
Gaia

Gaia framework aims at providing a generic computational environment to integrate physical spaces and their ubiquitous computing devices into a

programmable computing and communication system [5] [18]. It provides core services, including events, entity presence (devices, users and services), discovery and naming. By specifying well-defined interfaces to services, applications may be built in a generic way so that they are able to run in arbitrary active spaces.

Gaia's context infrastructure allows applications to obtain a variety of contextual information. Various components, called *Context Providers*, obtain context from either sensors or other data sources. These include sensors that track people's locations, room conditions (for example, temperature and sound) and weather conditions. *Context Providers* allow applications to query them for context information. Some *Context Providers* also have an event channel to asynchronously send context events. Thus, applications can either query a *Context Providers* or listen on the event channel to get context information.

All the ontologies in Gaia are maintained by an *Ontology Server*. Entities contact this server to get descriptions of other entities in the environment, information about context or definitions of various terms used in Gaia. The server also supports semantic queries to get, for example, the classification of individuals or subsumption of concepts. The *Ontology Server* also provides an interface for adding new concepts to existing ontologies.

Reasoning Approach. *Context Synthesizers* are Gaia components that collect context data from various *Context Providers*, derive higher level or abstract context from these lower-level context data and provide these inferred contexts to applications. Whenever a *Context Synthesizer* deduces a change in the inferred context, it publishes the new information. Gaia adopts two basic inference approaches [95]. *Rule-based Synthesizers* use pre-defined rules written in first order logic to infer different contexts. Each of the rules also has an associated priority, which is used to choose one rule when multiple rules are valid at the same time. However, if all the valid rules have the same priority, one of them is picked at random. Alternatively, some *Synthesizers* may use machine learning techniques, such as Bayesian learning and reinforcement learning, to infer high-level contexts. Past context information is used to train the learner.

3.3 OWL-SF

The distributed semantic service framework, OWL-SF [34], supports the design of ubiquitous context-aware systems considering both the distributed nature of context information and the heterogeneity of devices that provide

services and deliver context. It uses OWL to represent high-level context information in a semantically well-founded form, and its functional architecture integrates two basic building blocks: OWL Super Distributed Objects (OWL-SDOs) and Deduction Servers (DSs). OWL-SDOs are OWL enabled extensions of Super Distributed Objects (SDOs) [96], which encapsulate devices, sensors, user's interfaces (GUIs), services and other environmental entities and connect them to the upper context ontology, communicating using the Representational State Transfer Protocol [97]. Deduction Servers (DSs) are specific OWL-SDOs with an RDF inference mechanism and an OWL-DL reasoner. A system may be composed of multiple components of both types, which can be added and removed dynamically at runtime. DSs use the SDO discovery and announcement implementation to become aware of new SDOs in the environment. Whenever a new SDO is discovered, its semantic representation is added to the internal database.

Each SDO that encapsulates context providers and service-providing devices allows accessing the current state of an object as an OWL description. Each functional entity implemented as OWL-SDO has to be described using its own ontology containing terminological knowledge that enables the automatic classification of the object into appropriate service categories. The state of an object stores context values and is represented by an individual of a class in the ontology. Integrated reasoning facilities perform the automatic verification of the consistency of the provided service specifications and the represented context information, so that the system can detect and rule out faulty service descriptions and can provide reliable situation interpretation.

Reasoning Approach. Deduction servers (DSs) are specific OWL-SDO with an RDF inference mechanism and an OWL-DL reasoner. The rule-based reasoning process is provided by the RDF inference component and the deduced facts are used to trigger events to other SDOs and to process service calls. A subscription notification mechanism is used to monitor the SDO parameters to generate notifications whenever an observed parameter changes, triggering the deduction process to update the global ontology model accordingly. The RDF inference component is connected to the OWL-DL reasoner, which is responsible for classification and answering OWL-DL queries. The Racer system [71] is used as an OWL-DL reasoner. Besides providing deductive support, DSs are responsible for collecting the status of SDOs, published in the OWL format, and building an integrated OWL description accessible to the reasoning process. The semantic representation of each SDO is added to the internal database of the DS. This semantic representation consists of a set of instances

augmented with rules. Facts deduced from rules are only used to change parameters and to call services but never modify the knowledge base.

3.4 DRAGO

Distributed Reasoning Architecture for a Galaxy of Ontologies (DRAGO) is a distributed reasoning system implemented as a peer-to-peer architecture, in which every peer registers a set of ontologies and mappings [93]. In DRAGO, the reasoning operations are implemented using local reasoning over each registered ontology and by coordinating with other peers when local ontologies are semantically connected with the ontologies registered in other peers. DRAGO does not implement a context layer, i.e., it does not have any service for context collection, storing or distribution.

DRAGO is implemented to operate over HTTP and access ontologies and mappings published on the web. It aggregates a web of ontologies distributed amongst a peer-to-peer network in which each participant is called a *DRAGO Reasoning Peer* (DRP). A DRP is the basic element of the system and may contain a set of different ontologies describing specific domains of interest (for example, ontologies describing different activities of users in a university). These ontologies may differ from a subjective perspective and level of granularity. In a DRP there are also semantic mappings, each defining semantic relations between entities belonging to two different ontologies, described using C-OWL [98]. As these mappings establish a correlation between the local ontology and ontologies assigned to other DRPs, a DRP may also request reasoning services for other DRPs as part of a distributed reasoning task. Among the reasoning services DRAGO allows to check for ontology consistency, build classifications, verify concepts satisfiability and check entailment.

A DRP has two interfaces that can be invoked by users or applications. A *Registration Service Interface* is available for creating/modifying/deleting registrations of ontologies and mappings assigned to them. A *Reasoning Service Interface* enables requests of reasoning services for registered ontologies. To register an ontology at a peer the user specifies a logical identifier for it, i.e., a URI, and inform a physical location of the ontology in the web. Besides that, it is possible to assign semantic mappings to the ontology, providing, in the same manner, the location of the mappings on the web. New peers may be added dynamically to the system, providing new ontologies and semantic mappings. As each peer registers sets of heterogeneous ontologies and mappings, the knowledge base is totally distributed. When users or applications want to perform reasoning with a registered ontology they refer to the corresponding

peer and invoke its reasoning services giving the URI to which the ontology was bound.

Reasoning Approach. The reasoning process in DRAGO may compare concepts in different ontologies to check concept satisfiability, determining if a concept subsumes the other (i.e., the latter is less general than the former), based on the semantic mappings relating both ontologies. In a set of ontologies interconnected with semantic mappings, the inference of concept subsumption in one ontology (or between ontologies) may depend also on other ontologies related to the previous ones through those mappings. Every peer registers a set of ontologies and mappings, and provides reasoning services for ontologies with registered mappings. Each peer may also request reasoning services from other peers when their local ontologies are semantically connected (through a mapping) with the ontologies registered at the other peer. The reasoning with multiple ontologies is performed by a combination of local reasoning operations, internally executed in each peer for each distinct ontology. A distributed tableau algorithm is adopted for checking concept satisfiability in a set of interconnected ontologies by combining local (standard) tableaux procedures that check satisfiability inside the single ontology. Due to the limitations of the distributed tableau algorithm, DRAGO supports only three types of rules connecting atomic concepts in two different ontologies: *is equivalent*, *is subsumed* and *subsumes*. A Distributed Reasoner was implemented as an extension to the open source OWL reasoner Pellet [99].

3.5 P2P-DR

P2P-DR proposes a distributed solution for reasoning about context tailored to the special characteristics of AmI environments. This approach models the entities of an ubiquitous environment as nodes in a P2P system, in which each different node independently collects and processes the available context information. Specifically, it considers nodes that have exclusive knowledge, and that interact with neighbor nodes to exchange context information [28]. The knowledge of each node is expressed in terms of rules, and knowledge is imported from other nodes through bridging rules. As each peer may not have direct access to all sources of information, they share their knowledge through messages with their neighbor nodes. Moreover, the P2P-DR reasoning algorithm models and reasons with potential conflicts that may arise during the integration of the distributed knowledge.

Reasoning Approach. In a P2P-DR system, each peer has some computing and reasoning capabilities that it may use to solve a query about a local literal, based on its local and imported knowledge, which comprises context data and rules. A peer may not be able to solve the query locally, but it is aware of the knowledge that each of the other peers — that it can communicate with — possesses, and has mappings that define how part of this knowledge relates to its local knowledge. As each peer is willing to disclose and share its local knowledge, peers communicate with a subset of the other available peers to import the knowledge necessary to solve the query.

3.6 P2PIS

A peer-to-peer inference system (P2PIS [94]) is a network of peer theories. Each peer has a finite set of propositional formulas and can be semantically related by sharing variables with other peers. A shared variable between two peers is in the intersection of the vocabularies of the two peers. Not all the variables in common in the vocabularies of two peers have to be shared by them. Besides, two peers may not be aware of all the variables that they have in common but only of some of them. In a P2PIS, no peer has the knowledge of the global P2PIS theory. Each peer only knows its own local theory and the variables that it shares with some other peers of the P2PIS (its acquaintances). It does not necessarily knows all the variables that it has in common with other peers (including with its acquaintances). When a new peer joins a P2PIS it simply declares its acquaintances in the P2PIS, i.e., the peers it knows to be sharing variables with, and it declares the corresponding shared variables.

Reasoning Approach. In P2PIS the local theory of each peer is composed of a set of propositional clauses defined upon a set of propositional variables, called its local vocabulary. Each peer may share part of its vocabulary with some other peers. The system is capable of the reasoning task of finding consequences of a certain form (e.g., clauses involving only certain variables) for a given input formula expressed using the local vocabulary of a peer. Other reasoning tasks, e.g., finding implicants of a certain form for a given input formula, can be equivalently reduced to the consequence finding task. P2PIS distributed algorithm splits clauses if they share variables of several peers. Each piece of a split clause is then transmitted to the corresponding theory to find its consequences. The consequences that are found for each piece of split clause must then be re-composed to get the consequences of the clause that had been split.

3.7 Discussion

In this chapter, we discussed the need for distributed reasoning as a direct requirement that arises from the open, dynamic and heterogeneous nature of AmI. We described five proposals for distributed reasoning frameworks that support the deployment of AmI systems and try to overcome the limitation of traditional centralized reasoning approaches: Gaia, OWL-SF, DRAGO, P2P-DR and P2PIS. Among them, it may be said that Gaia and OWL-SF are the ones that best deal with dynamic scenarios, allowing context providers to be added or removed dynamically and ontologies to be dynamically modified with regard to types of context and their properties. Gaia offers a flexible and generic computational environment to fully implement and deploy AmI systems, and OWL-SF may be used for implementing such systems, despite not being tailored specifically for smart spaces, as its singular characteristic is its support for distributed inference. Both frameworks perform rule-based reasoning considering a distributed knowledge base. Moreover, they have two main advantages: (i) they allow event-based communication, so that a rule can be monitored and the result is sent for the subscriber when there is any change; and (ii) they allow the rules to be described with variables, in a more flexible way. In both frameworks, the aggregated context information in each reasoner will depend on the available providers, avoiding communication bottlenecks and allowing more efficient information processing and dissemination. However, the disadvantage of these approaches is that each reasoner is capable of reasoning only about the local context data, i.e., reasoners do not interact to exchange context information. As such, a context consumer has to know beforehand which context information will be available at each reasoner.

In contrast, DRAGO, P2P-DR and P2PIS propose distributed reasoning solutions considering data distributed over different elements in a AmI system. The main concern of DRAGO is to reason in distributed environments overcoming the barrier of the heterogeneous knowledge representation that independent entities in a AmI system are very likely to employ. As such, DRAGO is only capable of performing ontological reasoning to check concept satisfiability or subsumption. It relies on pre-defined mappings to align different ontologies. In a similar way, P2P-DR and P2PIS are peer-to-peer frameworks in which peers can communicate with a subset of the other available peers to import the knowledge necessary to answer queries based on mappings that define how its local knowledge relates to its peers knowledge. In such way, P2P-DR and P2PIS are capable of performing inference to answer queries that check if a rule is true or false, in which the knowledge, i.e., set of literals that represent

context information, is fully distributed in a peer-to-peer system. Nevertheless, P2P-DR and P2PIS are not capable of answering queries with variables. Moreover, DRAGO, P2P-DR and P2PIS are also limited by the fact that in practical implementations of AmI it is not feasible to build in advance mappings of all possible pairs of different ontologies that may be needed. Other techniques capable of dynamically aligning knowledge representations are more adequate in such conditions [74].

After discussing the positive and negative characteristics of the related work, we enumerate in the next section the design strategies for implementing a decentralized reasoning service and present our approach.