INTERFACING A REASONER WITH AN SDR: A PLATFORM AND DOMAIN API INDEPENDENT APPROACH

Jakub Moskal (Northeastern University, Boston, MA, USA; jmoskal@ece.neu.edu);
Mieczyslaw M. Kokar (Northeastern University, Boston, MA, USA;
mkokar@ece.neu.edu)

ABSTRACT

We present LiveKB, a framework for interaction between a reasoner and Software Defined Radio, which allows for use of dynamic facts that correspond to particular parameters within the SDR software, without having prior knowledge of its data structures. LiveKB is platform independent and domain API neutral, which allows it to sustain technological changes within a particular domain.

1. INTRODUCTION

Cognitive radio (CR) [1] allows for dynamic exploitation of contextual information that might come from different kinds of sources - RF environment, geographic location, waveforms, policies, device capabilities, status, etc. [2]. Several proposed approaches [2,3,4,5] to radio cognition are based on the Semantic Web technologies, namely on ontology and rule reasoning. The focus of this work is the problem of interfacing a CR's reasoner with Software Defined Radio (SDR) [6] to utilize its contextual data. The problem is presented in Figure 1. The reasoner, usually controlled by a monitor that is responsible for feeding the reasoner with facts and rules, needs to have access to the SDR software. SDR software is treated like a black box and the goal is to enable the reasoner to interact with the SDR regardless of its platform, its software model and without interfering with the end-user applications.

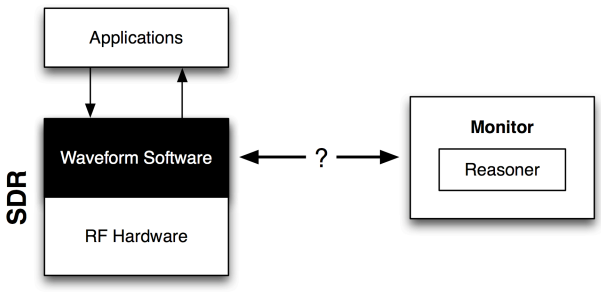

Figure 1. Interfacing Cognitive Radio reasoner with SDR

Two challenges arise when using knowledge-based reasoning in radios: 1) dealing with dynamically changing knowledge and 2) software platform independence between the reasoner and the radio's software. Using a knowledge base for reasoning over SDR's contextual data is challenging because its contents is partially stored on the radio itself, and thus is frequently updated, which might require non-monotonic reasoning or frequent restarting of the reasoner, which increases its runtime complexity. Secondly, there is no standard architecture for interfacing a CR's reasoner with SDR software components so that any reasoner that satisfies the architectural requirements could be used to monitor and control software that it has no prior knowledge about. We present an approach that tries to tackle both challenges with the use of Semantic Web technologies, like ontology and rules reasoning, and by the use of Common Object Requesting Broker Architecture (CORBA), a well-established standard within the SDR community.

2. RELATED WORK

One of the known approaches to CR, called Ontology-Based Radio (OBR) [3], utilizes architecture with internal buffers and custom messages passed between CR and its corresponding SDR. The architecture of OBR is shown in Figure 2. The Reasoning Component (RC) receives control messages from other radios and sometimes must update its contextual information and to modify SDR's parameters via Java reflection. An example implementation (in Java) was presented with a Prolog-based reasoner and OWL [7] ontologies converted into Prolog facts. This approach is platform-dependent because it requires SDR software to be written in Java.

The second approach, presented in [4,5], introduces a high-level Perception & Action Abstraction Layer (PAAL) which is responsible for mediating between CR and SDR, but requires SDR's API to be (most likely manually) wrapped to match PAAL. This solution is API-dependent because it will work only as long as the API is frozen. The abstract view of this interface is depicted in Figure 3. The PAAL layer is essentially a domain-specific API between

the SDR and the monitor. In this solution, the API becomes

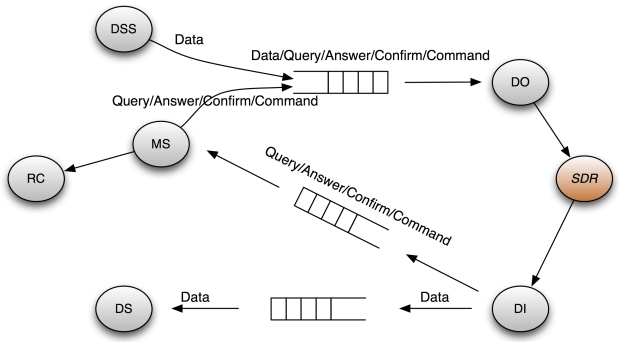

Figure 2. Architecture of Ontology-Based Radio [3]

the central part. It is reasonable to expect that there is going to be a need for introducing changes to such an API in order to match future technological advancements. Such changes would have to be followed by all SDR vendors that would like to conform to this API. Additionally, this framework does not address the problem of platform independence; the PAAL layer is intended to be platform-independent, but the authors do not elaborate on how this could be achieved.

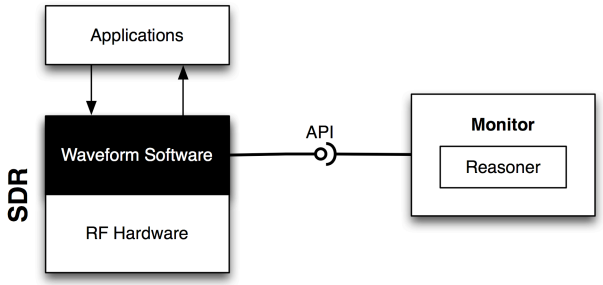

Figure 3. API-centered frameworks

2.1. Cognitive Radio without Semantic Web

There are a few interesting projects that take a different approach to cognitive radio and rather than augmenting radio's cognitive capabilities with the use of Semantic Web technologies, they focus on the software platform independence and the interoperability between different radios. Two of them are Software Communications Architecture (SCA) [8] from JTRS and a Cognitive Engine (CE) [9] from Virginia Tech.

2.1.1. Software Communications Architecture (SCA)

SCA is a software architecture framework that is becoming an industry standard for developing SDR's. SCA addresses the problem of waveform software portability by utilizing CORBA middleware, which handles the communication between software and the radio platforms. In this framework, waveform software and hardware components are described in XML, which SCA uses for loading, deploying and running required components on demand. There are two major benefits from using SCA – some separation between the waveform application and the radio platform [10], as well as software component distribution within the radio that is transparent to the software developers. This allows for running the same software on different hardware platforms. These benefits, however, come at an expense; developing SCA-compliant SDR's requires providing and maintaining extensive XML descriptions, which might be difficult without the support of additional software. Furthermore, SCA depends heavily on API's and before SDR becomes SCA-compliant it must go through a complex process of validation.

Despite the fact that it is a rather heavy-duty architecture, SCA seems to be an attractive approach for developing portable software for SDR's. Nevertheless, it is not a standard that was designed with radio cognition in mind. SCA doesn't include an API for reasoners. Therefore it is not a ready solution for Cognitive Radios.

2.1.2. Cognitive Engine (CE)

Similarly to the previously described PAAL layer, the heart of Cognitive Engine architecture is an API that allows for interfacing a cognitive engine with an SDR. The SDR itself is viewed as a set of *knobs and meters,* which are described in a XML document. The CE itself is a set of genetic algorithms that make use of the SDR's parameters in order to achieve interoperability between different radios. Adopting this framework requires implementing a specific API, which exhibits the same problems as PAAL, where the API becomes the bottleneck of the framework.

3. LIVEKB

LiveKB is a framework that extends the architecture of OBR to allow a platform-independent access to SDR's software parameters. Figure 4 shows how LiveKB fits in with OBR. Similarly as SCA, LiveKB achieves platform-independence through the use of CORBA middleware. SDR software objects must be registered with CORBA's Naming Service, which allows for locating distributed objects by looking up their logical names, as opposed to their actual references. Once LiveKB finds SDR's objects, it allows the reasoner to monitor and manipulate them according to the domain rules. The reasoner interacts with the SDR software using ontological terms, which are platform-independent, thus SDR can be implemented in any language that supports CORBA.

LiveKB is suitable to work with any SDR that registers its objects with CORBA Naming Service, but it needs the information about the SDR's software model so that it can

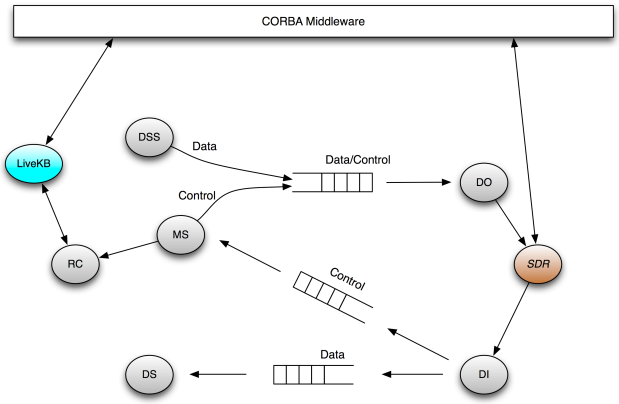

Figure 4. LiveKB as an extension of OBR

find and manipulate these objects properly. CORBA uses Interface Definition Language (IDL) for defining interfaces of software components, independently of what language the components are implemented (e.g., C++ or Java). IDL, which is our choice for describing software models within the LiveKB framework, could be substituted with a different language that allows for easy serialization and manipulation, e.g. UML and XMI.

3.1. ABox and TBox

LiveKB passes information about objects registered with CORBA Naming Service to the reasoner in the form of OWL ontologies. It is important to make a distinction between two kinds of knowledge that is passed to the reasoner. The semantics of OWL is based on *Description Logics (DL)*, a family of Knowledge Representation (KR) formalisms equipped with a formal, logic-based semantics [11]. Figure 5 presents architecture of a system based on DL. Such a system allows for setting up a Knowledge Base (KB) and for reasoning about its content. KB consists of two components, a *TBox* and an *ABox*. In the context of OWL, the TBox introduces axioms defining the ontology – classes and properties; the ABox contains assertions about named individuals – instances of classes and properties defined in the TBox.

In the context of LiveKB, the software model, given to LiveKB in the IDL form and converted into OWL, stands for the TBox component and allows the reasoner to understand the structure of the model. The ABox part of the KB must be collected in the runtime and corresponds to the current values of parameters in the radio.

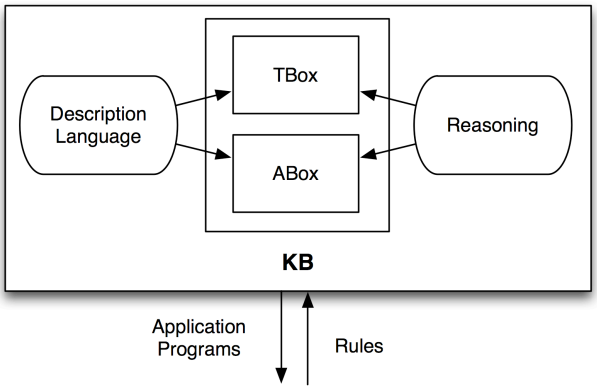

Figure 5. Architecture of a knowledge representation system based on Description Logics [11]

When the reasoner is provided with both a TBox and an ABox, it can reason over the knowledge base that corresponds to the current contextual data stored in the runtime objects of the SDR. Domain experts can write rules based on the terms defined in the OWL ontology without the need of knowing the platform or the API to access the SDR's internal parameters. Thus the rules can take advantage of the contextual information and act accordingly, e.g. update software to improve the QoS.

3.2. LiveKB architecture and workflow

The architecture of LiveKB is presented in Figure 6. All the interaction between the reasoner and the radio is handled by CORBA with no domain-specific API in use. LiveKB works in two different phases – *offline* and *online*. The offline phase consists of two steps – 1) retrieving and compiling the IDL definitions of the SDR software into the LiveKB's implementation language stubs and 2) converting the IDL into an OWL TBox representation. Both steps can be performed automatically. This phase is done only once, each time the application is started. The online phase is responsible for utilizing the products of the offline phase to create ABox facts that represent the current state of the SDR's objects. This phase might occur either periodically to reflect the changes in the software or on demand, driven by the reasoner.

The interaction between LiveKB and CORBA is supported by the use of the CORBA-OWL Mapper, which maps CORBA IDL definitions into ontological terms and vice versa. This mapping needs to be performed manually if names of interfaces and attributes in IDL don't correspond directly to names of classes and properties in OWL.

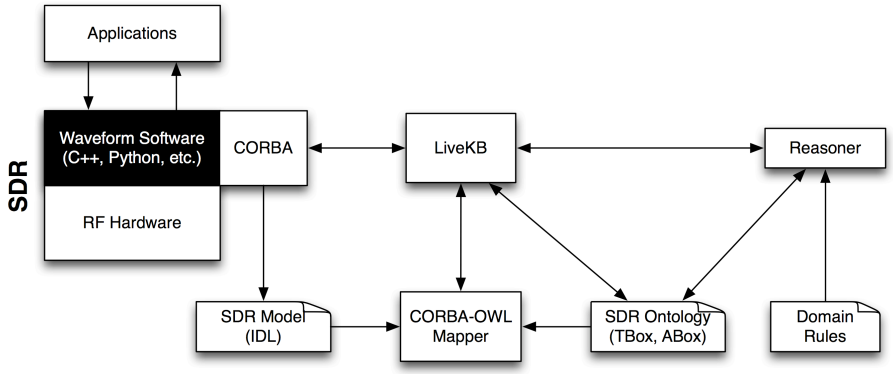

Figure 6. Architecture of LiveKB

It is worth mentioning that LiveKB could be implemented in any language that supports reflection. Our prototype implementation was based on Java, thus IDL definitions were used to generate Java stubs. The reflection on SDR's runtime objects is performed according to the information included in the TBox. Once the ABox is generated in the online phase, the entire KB is passed to the reasoner, which can now process rules written by domain experts. The reasoner does not need to be aware of the SDR's structure and platform.

To take advantage of the runtime information, the reasoner must be able not only to monitor, but also to update the SDR's parameters. Similarly to creating the ABox, the monitoring and the updating must be performed via reflection on the IDL-generated stubs. Through the use of a domain-independent API, LiveKB provides this functionality to the reasoner. LiveKB performs the update on the CORBA objects, creates a new ABox with updated values and provides the ABox to the reasoner.

3.3. Challenges with dynamic change of the knowledge base

Changing the values of parameters in the runtime requires frequent updates to the knowledge base, which as described in [12], poses a challenge in OWL-based inference, particularly because OWL is monotonic. In monotonic logics facts can only be added, but not updated. One way to deal with this problem is to restart the reasoner with a new ABox each time a change is made, or in a periodical fashion, however, this may not prove to be efficient enough.

4. CURRENT STATE OF THE IMPLEMENTATION

Current implementation of the framework is based on a number of assumptions and simplifications. First of all, CORBA IDL definitions are constrained only to interfaces and attributes. In our experiments, IDL definitions were manually converted into OWL. Interfaces in IDL became classes; attributes of primitive types became datatype properties in OWL. Attributes, which were instances of other interfaces in IDL, were converted into object properties in OWL. Moreover, to simplify the implementation, we assumed a tree structure of the model, i.e., each object in the API could be accessed via reflection on a *root* object. The structure (being a tree) did not have any reference cycles. The names of CORBA attributes were unique in the global scope and corresponded directly to the names of properties in the OWL ontology.

As mentioned earlier, LiveKB was implemented in Java and it was interfaced directly with only one reasoner, BaseVISor [13]. BaseVISor can perform inference with a subset of OWL axioms and over rules, which suited our needs. Additionally, BaseVISor is implemented in Java, which provides reflection, necessary for updating parameters. Each update was not reflected in the knowledge base until the next run of the LiveKB.

5. CONCLUSIONS AND FUTURE WORK

We presented LiveKB, a framework that addressed the problem of interfacing a reasoner with a SDR in a platform and domain API independent fashion. The platform independence was achieved with the use of CORBA; the problem of domain API independence was solved by interfacing the reasoner with the SDR software based on the information contained within the IDL definitions of the model.

The LiveKB framework depends on CORBA middleware, which may exhibit problems in the radio domain [14]. Even though CORBA could be replaced with a different mechanism that allows objects lookup and control in a language-neutral fashion, CORBA seems to be an accepted framework in the embedded community, cf. CORBA implementations on DSP's or directly in FPGA's [15].

As described in the paper, the current implementation is not complete and relies on a number of assumptions and simplifications. Augmenting the implementation to fit the architecture is going to be a major part of the future work. We plan to test this framework with a real-world application and with the use of available SDR's, cf. GNU Radio. As mentioned earlier, we intend to develop a generic API between the LiveKB and a reasoner, so that different reasoners could be used.

6. REFERENCES

[1] J. Mitola III and G. Maguire, "Cognitive Radio: Making Software Radios More Personal", *IEEE Personal Commun. Mag.*, pp.13—18, Aug 1999

[2] J.D. Poston, W.D. Horne, M.G. Taylor and F.Z. Zhu, "Ontology-Based Reasoning for Context-Aware Radios: Insights and Findings from Prototype Development", *New Frontiers in Dynamic Spectrum Access Networks, DySPAN 2005*, pp.634—637, 8—11 Nov 2005

[3] J. Wang, M.M. Kokar, K. Baclawski and D. Brady, "Achieving self-awareness of SDR nodes through ontology-based reasoning and reflection", *Software Defined Radio Technical Conference SDR'04*

[4] A. Ginsberg, J.D. Poston and W.D. Horne, "Experiments in Cognitive Radio and Dynamic Spectrum Access using An Ontology-Rule Hybrid Architecture", *Second International RuleML-2006 Conference*, 2006

[5] A. Ginsberg, W.D. Horne and J.D. Poston, "Community-Based Cognitive Radio Architecture: Policy-Compliant Innovation via the Semantic Web", *New Frontiers in Dynamic Spectrum Access Networks, DySPAN 2007,* pp.191—201, 17—20 Apr 2007

[6] J. Mitola III, "The Software Defined Radio Architecture", *IEEE Commun. Mag.,* vol. 33, no. 5, pp. 26—38, May 1995.

[7] W3C, "Web Ontology Language (OWL)", http://www.w3.org/2004/OWL/

[8] Joint Program Executive Office (JPEO) JTRS, "Software Communications Architecture (SCA) v.2.2.2", http://sca.jpeojtrs.mil/

[9] D. Scaperoth, B. Le, T. Rondeau, D. Maldonado and C. Bostian, "Cognitive Radio Platform Development for Interoperability", *Military Communications Conference MILCOM 2006,* pp.1—6, 23—25 Oct 2006

[10] G. Gailliard, E. Nicollet, M. Sarlotte and F. Verdier, "Transaction level modeling of SCA compliant software defined radio waveforms and platforms PIM/PSM", *In Proceedings of the Conference on Design, Automation and Test in Europe,* Nice, France, 16—20 Apr 2007

[11] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation, Applications,* Cambridge University Press, Cambridge, UK, 2003

[12] M.M. Kokar and L. Lechowicz, "Language Issues for Cognitive Radio", *Proceedings of the IEEE*, vol. 97, no. 4, pp. 689—707, Apr 2009

[13] C. Matheus, K. Baclawski and M.M. Kokar, "BaseVISor: A Triples-Based Inference Engine Outfitted to Process RuleML and R-Entailment Rules", *In Proceedings of the 2nd International Conference on Rules and Rule Languages for the Semantic Web*, Athens, GA, Nov 2006

[14] F. LeRoy, G. Abgrall, J. Delahaye, J. Diguet, G. Gogniat, "A Comparative Study of Two Software Defined Radio Platforms", *Software Defined Radio Technical Conference SDR'08*

[15] F. Casalino, G.Middioni and D.Paniscotti, "Experience Report on the use of CORBA as the sole middleware solution in SCA-based SDR Environments", *Software Defined Radio Technical Conference SDR'08*