UNIVERSITY OF CALIFORNIA,
IRVINE

COAST Services: Achieving Service Customization and Policy-Based Differential Access in Personal Information Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Sciences

by

Alegría Baquero Merino

Dissertation Committee:
Professor Richard N. Taylor, Chair
Professor André van der Hoek
Professor Michael J. Carey

2014

DEDICATION

To Reuben and Oliver, the loves of my life.

To my parents, my everlasting source of love, support, and encouragement.

TABLE OF CONTENTS

Page

LIST OF FIGURES **vi**

LIST OF TABLES **viii**

ACKNOWLEDGMENTS **ix**

CURRICULUM VITAE **xi**

ABSTRACT OF THE DISSERTATION **xiv**

1 Introduction: Service Security and Customization **1**

2 Motivation **7**
2.1 Motivating Domain: EHR Management 9
2.1.1 Motivating Scenarios: Obtaining EHR Data 12

3 Research Questions and Goals **15**

4 The Context of this Work **18**
4.1 Software Architecture 18
4.2 Decentralized Systems 20
4.3 Information Management Systems 21
4.4 Web Services 22
4.4.1 Service Provision 22
4.4.2 Service Composition 23
4.4.3 Service Customization 25
4.5 Service Security and Access Control 26
4.6 Privacy 28
4.6.1 Privacy Concerns in the Healthcare Domain 29
4.7 Privacy Policy Languages 31
4.8 Policies and Services 33
4.9 Trust and Reputation 36
4.10 Ontologies 37
4.10.1 Healthcare Information Standards 39

5 Foundational Technologies: COAST and Policy Languages 42
5.1 The COAST Architectural Style . . . 43
5.1.1 Architectural Foundations . . . 43
5.1.2 Motile/Island . . . 47
5.1.3 Comparison with other styles and technologies . . . 49
5.2 Policy Specification Languages . . . 51
5.2.1 Evaluation Criteria . . . 52
5.2.2 Languages Overview and Evaluation . . . 54
5.2.3 Evaluation summary . . . 68

6 Policy-Based COAST Services 72
6.1 Policy-based Differential Service Provision . . . 75
6.1.1 Specifying Policies . . . 75
6.1.2 Policy Evaluation . . . 83
6.1.3 Associations of Policy and Service Capabilities . . . 85
6.1.4 Service CURLs . . . 87
6.1.5 Services Definition . . . 89
6.1.6 Dynamic Creation of Consumer-specific Services . . . 91
6.1.7 Capability Accounting . . . 96
6.2 Consumer-controlled Service Customization . . . 99
6.2.1 Customizing Single-Source Services . . . 100
6.2.2 Customizing Multi-Source Services . . . 103
6.2.3 Comparison to Other Related Technologies . . . 107
6.3 Limitations and Scope of this Work . . . 110

7 Practical Experiments: COASTmed 114
7.1 Empirical Domain: EHR Management . . . 115
7.1.1 Rethinking the EHR Scenario: Sharing and Using Patient Data in COASTmed . . . 117
7.1.2 EHR Data Model . . . 119
7.2 Initial System Architecture . . . 120
7.3 Specifying Policies in COASTmed . . . 123
7.3.1 Describing Policies Using Ontologies and User Interfaces . . . 127
7.4 Generating Service CURLs . . . 129
7.4.1 Evaluating Policies at CURL Creation Time . . . 130
7.4.2 Capability Accounting . . . 135
7.5 Using Services through User-specific CURLs . . . 136
7.5.1 Evaluating Policies at Service Use Time . . . 136
7.5.2 Other Explored Techniques to Retrieve Facts . . . 139
7.5.3 Dynamically Creating User Services . . . 141
7.5.4 Typed Messages . . . 143
7.5.5 Capability Accounting for Tracking Service Usage . . . 144
7.6 COASTmed within a Large Organization . . . 146
7.7 Implementation Technologies . . . 148
7.8 Design and Development Experience Insights . . . 150

8 Evaluation **153**
8.1 Scenario-based Evaluations . . . 154
8.1.1 Simulation Setup . . . 156
8.1.2 Goal 1: Differential Access Support . . . 157
8.1.3 Goal 2: Capability Revocation . . . 163
8.1.4 Goal 3: Service Customization . . . 169
8.1.5 Goal 4: Data Integration . . . 171
8.2 Comparative Analysis . . . 178
8.2.1 Technologies Overview . . . 178
8.2.2 Systems Overview . . . 185
8.2.3 Analysis Results . . . 210
8.2.4 Discussion . . . 217
8.3 Generalizability to Other Domains . . . 221

9 Conclusion **223**

Bibliography **231**

A Examples of privacy policies **252**
A.0.1 Hospital organizational privacy policies . . . 252
A.0.2 Insurance company organizational privacy policies . . . 257

B Sample healthcare policies expressed in various policy languages **259**
B.1 EPAL . . . 260
B.2 XACML . . . 278
B.3 Cassandra . . . 306
B.4 PeerTrust . . . 308
B.5 Ponder . . . 310
B.6 Rei . . . 313

C PrimaCare's data model **316**

LIST OF FIGURES

Page

2.1 A patient in the effort to access and share his EHR. 13
2.2 Researchers trying to obtain and organized distributed medical information. 13
2.3 Patient trying to obtain data from multiple, inter-related service providers. . 14

5.1 Notional structure of a COAST execution host [121]. 45

6.1 A provider, based on a set of policies, dynamically creates user-specific services. 74
6.2 A policy's contents. 76
6.3 Associations between policies and capabilities. 86
6.4 Explicit associations of policies and capabilities. 87
6.5 A CURL's anatomy. 88
6.6 Obtaining a user-specific CURL. 89
6.7 A provider's services. 90
6.8 A healthcare provider's services. 91
6.9 Sequence of messages for CURL request, and service creation and use. 94
6.10 Dynamic creation of user services. 95
6.11 Valid and revoked CURL scenarios. 97
6.12 Service customization. 100
6.13 A COAST peer running 3 user services. 101
6.14 EHR service customization. 102
6.15 Independent star topology. 104
6.16 Dependent star topology. 104
6.17 Ring topology. 105
6.18 Independent star topology scenario. 105
6.19 Dependent star topology scenario. 106
6.20 Ring topology scenario. 107

7.1 Parties and interactions in the healthcare domain. 116
7.2 Initial system architecture. 121
7.3 System architecture at t_n. 122
7.4 A relational database structure to store policies 125
7.5 User interface to specify policies 126
7.6 Policy creation process 127
7.7 Using ontologies for policy specification 128
7.8 A service consumer's service CURL request 129

7.9 CURL sent to Dr. Smith 134
7.10 Dynamic user-specific service creation 137
7.11 Dynamic service creation based on CURL's data 142
7.12 A master policy maker creating a new global policy within each service provider system. 147
7.13 Policies also being retrieved from a global database at CURL creation time. . 148

8.1 Simulation's main screen. 156
8.2 Scenario 1a: Jim (UUID 1) viewing John's EHR (UUID 5) 159
8.3 Scenario 1a counterexample: Jim (UUID 1) attempting to update Jane's EHR (UUID 6) 159
8.4 Scenario 1b: Jackie (UUID 2) viewing John's EHR (UUID 5) 161
8.5 Scenario 1b counterexample: Jackie (UUID 2) attempting to view Jane's EHR (UUID 6) 161
8.6 Scenario 1c: Tom (UUID 3) viewing John's EHR (UUID 5) sometime between 9AM and 5PM. 163
8.7 Scenario 1c counterexample: Tom (UUID 3) is denied to access John's EHR (UUID 5) after 5PM. 163
8.8 Scenario 2a: Laura (UUID 4), a hospital administrator viewing the organization's payrolls at t_0. 164
8.9 Scenario 2a counterexample: Laura (UUID 4), at t_1, receives an error message when attempting to use again the view-payrolls capability. 165
8.10 Scenario 2b t_0: Mary (UUID 8), a hospital resident accessing John's EHR (UUID 5). 166
8.11 Scenario 2b t_1: John (UUID 5) forbids Mary (UUID 8) from accessing his EHR. 167
8.12 Scenario 2b t_3: Mary (UUID 8) no longer can access John's EHR (UUID 5). 167
8.13 Scenario 2c: Tom (UUID 3) is denied the access to Jane's EHR (UUID 6). . 168
8.14 Scenario 2c counterexample: Tom (UUID 3) can successfully access John's EHR (UUID 5). 168
8.15 Scenario 3: Jane (UUID 6) composes a custom closure to obtain an specialist referral from the hospital an make with it an appointment with the specialist. 171
8.16 Goal 4, scenario 2 input and output for CDC. 174
8.17 Goal 4, scenario 1 input and output for patient John. 177

LIST OF TABLES

Page

5.1 Privacy policies in the healthcare domain 53
5.2 Evaluation privacy policies languages 70

7.1 Sample policies 131
7.2 Evaluation of sample policies 133
7.3 Sample policies and capabilities associations 134
7.4 Differences between first and second stage of evaluation 140

8.1 Sample users 157
8.2 ALDSP summary 187
8.3 X-GTRBAC + WS-Policy summary 190
8.4 Marmite summary 192
8.5 Cassandra summary 194
8.6 Haas et al. summary 196
8.7 InfoShare summary 198
8.8 Privacy Policy Compliance System (PPCS) summary 199
8.9 Coetzee et al. Web Service access control summary 202
8.10 OASIS summary 205
8.11 PERMIS PMI summary 207
8.12 Privacy and Access Control for IHE-Based Systems summary 210
8.13 Evaluation summary 217

ACKNOWLEDGMENTS

It is very hard to succeed without the support, encouragement, and love of people around you. I am incredibly lucky to have had in this long road people close to me who gave me that and more, and to whom I owe this dissertation and the achievement of finishing my Ph.D. I dedicate this dissertation to my loving husband Reuben, to my son Oliver, and to my parents Ximena and Xavier.

Reuben, you have supported and encouraged me all the way since the day we met, a few days after I arrived to UCI. You have always made me feel a better person, student, and professional than I am. It has been your faith in me that has constantly pushed me forward. You are an example of perseverance, resilience, and hard work. I thank everyday and praise my luck to have you by my side.

Oliver sweetheart, your smile is my daily reward to all the difficulties and the hard work. You keep me going and give me more reason to succeed and be better everyday. My love, never stop learning and being curious, there is so so much to learn and discover in this world. Enthusiastically pursue goals and dreams that will make you proud of yourself, just like mommy feels today. If we were all curious and creative like toddlers are, science would be far more advanced by now!

Mami y papi, you have always been there beside me, supporting me in every way to follow any dream I have ever had. Thanks to you I have lived experiences, gone places, and done things that without you it would have never been possible. I can't thank you enough for everything. Thank you for taking care of Oliver and giving him so much love in this final stage of the dissertation which was key in finishing up. I cannot thank you all of you enough, I love you so much and I hope I have made you proud.

Thank you so much Professor Taylor for taking me as your student and for all these years of support. I didn't really know what a Ph.D. was, let alone imagined being a Ph.D. myself—I owe this to you. Joining the lineage of great students such as Roy Fielding, Nenad Medvidovic, Peyman Oreizy, Jim Whitehead, Rohit Khare, Girish Suryanarayana, Eric Dashofy, Justin Erenkrantz, Scott Hendrickson, and many others is a great honor to me.

Many thanks to my advancement, topic proposal and dissertation committee: Dr. André van der Hoek, Dr. Michael Carey, Dr. Alfred Kobsa and Dr. Sarah Lawsky. Your support and valuable feedback have been crucial to finishing my degree. A very special thanks to André; you have always given me words of encouragement in moments when I needed them the most, motivating me to continue and to feel more deserving of this degree.

Thanks to Yunan Chen who helped us understand more about the healthcare domain, a new area for us, where we hope to have contributed with our research. Thanks to Dr. Ban Al-Ani and Dr. David Kay for the great learning opportunity while being their teacher assistant.

A special thanks to Allan Schiffman and Justin Erenkrantz who took a chance on me, allowing me to work with them during the summer (CommerceNet in 2012 and Bloomberg in 2014).

These were enriching opportunities that not only let me grow as a professional and learn so much, but gave me the opportunity to meet wonderful people and colleagues with whom I have kept in contact. Thank you Allan and Justin for the invaluable chance you gave me and for your friendship.

Thank you to my colleagues in the COAST group. To Kyle Strasser for all your patient explanations. To Michael Gorlick, thanks to whom I realized I am stronger than I thought, and that perseverance and determination can triumph over anything.

Thanks to Kris Bolcer and Gina Anzivino (formerly) at the ICS Student Affairs office, our Department Manager Marty Beach, and to Debi Brodbeck at the Institute for Software Research who are always supporting and helping us navigate the system. They have always been incredibly nice and helpful. Our academic lives would be difficult without them.

To my dear family and family-in-law, who are always eager to know about my work, constantly encouraging me, and wishing me success. Special thanks to my uncle Dennis, my aunt Patty, Mimí, and to my mother and father-in-law Lalah and Roger. Throughout these years you have always showed an interest, constantly cheering for me, and giving me good vibes along the way.

Thankfully, I have not only been buried in papers, but have had a great time during grad school. Thank you my dear friends for the good times, the conversations, the trips, the laughs, the parties, and for much more. Thanks to Ruy C., Nobu T., Jam R., Nithya S., Kristina W., Ramzi N., Leyna Z., Yongjie Z., Tijana M., Vesna M., Kenny D., Kah L., Joel O., Ashley O., Nick M., Nicolás L., Paco S., Anupa M., Max M., Tiago P., Marisa C., Dan W., Chris F., Megan B., Peter M., Kevin D., Liz. C., Connie M., Eduardo M., Michelle P., Jasmine K., Kurt B., Nathanael M., Megan T., Kristen S., Kristine G., Ines V., Erin E., Arjun S., Michael B., Sergi P., Stephy B., and Alex B.

I want to specially thank my colleague and friend Rosalva Gallardo who has supported me professionally and personally, helped me prepare for big milestones, and always gave me good advice. Grad school would not have been the same without a friend like you.

Also, I want to thank my dear friend Julia West who passed away in March 2013. Our paths' crossing changed my life forever. Thanks to her, I met my husband Reuben on October 2007. I am eternally grateful to Julia.

Thanks to The National Science Foundation and to Bloomberg L.P. for supporting our research. Thanks to IEEE, ACM, and Springer for publishing our work.

Thank you, University of California, Irvine for this wonderful experience and for preparing me for what is to come.

The text of this thesis/dissertation is a reprint of the material as it appears in "Secure and Customizable EHR Management Services with COASTmed". The co-author listed in this publication directed and supervised research which forms the basis for the thesis/dissertation.

CURRICULUM VITAE

Alegría Baquero Merino

EDUCATION

Doctor of Philosophy in Information and Computer Sciences **2014**
University of California, Irvine *Irvine, California*

Master of Science in Information and Computer Sciences **2009**
University of California, Irvine *Irvine, California*

Marketing Engineering Degree **2001**
Universidad Internacional del Ecuador *Quito, Ecuador*

Laurea in Computer Graphics **1999**
Istituto Europeo de Design *Milan, Italy*

RESEARCH EXPERIENCE

Graduate Student Researcher **2009–2014**
Institute for Software Research, University of California Irvine *Irvine, California*

Summer Intern **2014**
Bloomberg L.P. *New York, New York*

Summer Intern **2012**
CommerceNet *Palo Alto, California*

TEACHING EXPERIENCE

Teaching Assistant **2011**
Software Architecture and Distributed Systems, Prof. Ban Al-Ani
University of California, Irvine *Irvine, California*

Teaching Assistant **2011**
Human Computer Interaction, Prof. Alfred Kobsa
University of California, Irvine *Irvine, California*

Teaching Assistant **2010**
Informatics Core Course I (Scheme), Prof. David Kay
University of California, Irvine *Irvine, California*

REFEREED CONFERENCE PUBLICATIONS

Secure and Customizable EHR Management Services with COASTmed **Jul 2014**
Symposium on Foundations of Health Information Engineering and Systems and the Software Engineering in Healthcare (FHIES/SEHC 2014), Washington, DC

COASTmed: Software Architectures for Delivering Customizable, Policy-Based Differential Web Services **May 2014**
36th International Conference on Software Engineering (ICSE 2014), Doctoral Symposium, Hyderabad, India

Blend me in: Privacy-Preserving Input Generalization for Personalized Online Information Services **Jul 2013**
11th annual Conference on Privacy, Security and Trust (PST 2013), Tarragona, Catalonia

Computational Commerce: a Vision for the Future **Sep 2012**
13th International Conference on Electronic Commerce and Web Technologies (EC-Web 2012), Vienna, Austria

POSTERS AND EXTENDED ABSTRACTS

Enabling Secure, Customizable, and Dynamic Personal Information Services with COAST **Oct 2014**
Grace Hopper Celebration Of Women in Computing, Poster session, Phoenix, Arizona

CREST: principled foundations for decentralized systems **Oct 2011**
ACM International Conference on Object Oriented Programming, Systems Languages and Applications, Portland, Oregon

HONORS AND AWARDS

Grace Hopper Celebration of Women in Computing Scholarship Grant **2014**

Conference Presentation Grant (USD 275) **2014**
Donald Bren School of Information and Computer Sciences, UC Irvine

NSF travel grant, ICSE 2014, Hyderabad, India (USD 1,448) **2014**

ACM-W Scholarship Award (USD 1,200) **2014**

The Coalition to Diversify Computing (USD 1,000) **2014**

Google Anita Borg Scholarship Finalist (USD 1,000) **2011**

CRA-W Graduate Cohort (CRA-W) Scholarship **2009 & 2010**

Miguel Velez Fellowship (USD 10,000) **2009**
University of California, Irvine

ICS Fellowship **2009**
University of California, Irvine

Cum Lauder Honor **2002**
Universidad Internacional del Ecuador

SERVICE ACTIVITIES

Student volunteer **May 2014**
International Conference on Software Engineering, Hyderabad, India.

Student volunteer **May 2011**
International Conference on Software Engineering, Honolulu, Hawaii.

Co-organizer of the ICS Graduate Student Seminar **2010-2011**
University of California, Irvine]

ABSTRACT OF THE DISSERTATION

COAST Services: Achieving Service Customization and Policy-Based Differential Access in Personal Information Systems

By

Alegría Baquero Merino

Doctor of Philosophy in Information and Computer Sciences

University of California, Irvine, 2014

Professor Richard N. Taylor, Chair

People and organizations constantly exchange personal information such as health data. However, the use and exchange of this information poses two salient challenges. First, trust among data users and providers is not homogenous, but data is supplied according to individual authority and rights. Second, people and organizations use this data for uncountable and often divergent purposes. State-of-the-art web services are rigid, "one-size-fits-all" solutions that do not meet all users' needs nor allow providers to distinguish among users. This tension between information need and service provision calls for sophisticated mechanisms to simultaneously enable customization and service access based on specific trust relationships. Our goals are twofold: first, enable differential access to a provider's services—data and computation capability—according to privacy and operational policies. Second, enable consumer-controlled service customization to access and computationally manipulate data to fulfill specific needs within the authority granted by the provider. Our approach leverages the COAST architectural style's principles and implementation mechanisms and the Rei policy language. The context of our work is decentralized information systems, where constituent personal services operate under multiple, distinct authorities. We evaluate our approach in the context of the healthcare domain and present COASTmed, an EHR management system prototype which exhibits the proposed solutions to the described challenges.

Chapter 1

Introduction: Service Security and Customization

In the process of routine interactions with different organizations, people disclose personal information with the purpose of obtaining some service, supporting some group, or simply because it is required by law. As a result, personal information is distributed among many agencies: travel services hold holiday itineraries, financial agencies manage accounts and loans, medical providers store health records, schools track for academic performance, and so on. Personal information is used and shared by individuals and organizations according to nuanced trust relationships in a wide array of domains for various purposes.

However, sharing and use of individuals' data in decentralized contexts presents ***two salient problems***. First, the dissemination of personal data poses privacy concerns. Individuals and organizations spontaneously restrict access to their information due to lack of trust or just because some party has no business accessing certain kinds of data. However, it is difficult to capture these heterogeneous trust relationships among consumers and providers of personal information—both individuals and organizations—in software systems and support nuanced,

per-user access permissions to personal information services. As a result, these services' capabilities are often reduced to the "minimum common denominator", restricting access to privileged users that would otherwise have extended access authority to an organization's assets. In addition, insiders' abuse within organizations is not uncommon, and neither is the incongruence between data disclosure policies and offered services. Hence organizations lack sufficient guarantees of privacy policy compliance.

Second, the myriad of current and potential uses of personal information in an array of domains makes it difficult for information providers—people or organizations—to personalize their services to meet the specific needs of all users. Therefore, services are usually rigid "one-size-fits-all" solutions that hardly satisfy any of their users' needs and render tasks more difficult—services may return additional unrequested data and require considerable post-collection manipulation. It is a recurrent problem for system developers, for example, to find suitable web services for their application needs. Current web services follow a "vending machine model" where functions are unilaterally made available online and users have no control or flexibility to customize when interacting with these services [133], therefore creating a tension between information need and service provision, and diminishing the overall utility of the service.

Given the described problems, the ***main challenges*** are (a) granting access to personal information according to appropriate, desired relationships between individuals and organizations (e.g., granting a doctor access to a patient's health records, but not to his financial data), and (b) allow individuals and organizations to obtain the required personal information for diverse, specific uses (e.g., allowing the doctor to run an algorithm using a patient's blood tests result held at a laboratory so that he can infer the cause of symptoms).

To approach these challenges and ameliorate the described problems, novel mechanisms are needed to build software systems which (a) allow service providers to offer flexible, fine-grained services (arrangements of capabilities) on a per-user basis and according to privacy

and operational policies, and; (b) enable service consumers to customize services to suit specific information and computational needs.

Current technologies fail to adequately meet the described challenges typically occurring in highly decentralized settings. Despite research and development of authorization models for controlled data access, web services aimed for data integration, and customization efforts (section 4), adequate solutions offering differential authority and customization of personal information services have not yet been realized.

Our ***research goals*** (detailed in section 3) are twofold:

- First, enable differential access to a provider's services, where the provider has the capacity to distinguish among service consumers, making available personal information and computational capability according to privacy and operational policies. Differential service provision permits organizations treating users idiosyncratically, maintaining multiple trust relationships, and restricting and expanding the per-user availability of information and service functionality.
- Second, enable service customization, allowing service consumers to manipulate personal information and use the services' computational capabilities at will, however within the given authority and rights to access these assets.

These goals address clear shortcomings in the current practice, which is often fraught with bureaucratic and inefficient (often manual) intra- and inter-organizational processes for sharing, using, and protecting personal information, burdening all the involved parties:

- *Individuals* encounter difficulty in accessing, in a meaningful and automated way, their own data when it is spread across many agencies. Similarly, they are often unable to control the dissemination of their personal data, in contrast to specifically authorizing parties to manipulate particular data.

- *Data holders* have difficulty in providing particularized access to individuals' data when access rights and trust relationships are dynamic and complex; often the de facto outcome is loosely defined and coarse-grained boundaries.

- *Service providers* are constrained to offer a few standard services to their users due to the overhead of attempting to suit services to fit all individual needs. Services typically remain static—despite needs for change arising—because of the potential for inadvertently disrupting existing users. Moreover, the services provided are often divorced from privacy and operational policies, rendering vulnerable the privacy of the information, and providing opportunity for the discrepancy among drifting services and policies, and for insiders authority abuse.

- *Service users*—even when authorized—have difficulty accessing personal information when it is spread across various data holders and held in different formats, making it difficult to integrate and manipulate the data to fit specific needs.

The ubiquity of web services, the popularity of mashups, continuous customization and personalization efforts, the vast dissemination of individuals' data, widely adopted data mining practices, increasing security threats to personal data, and the proliferation of integration standards are indicators that our goals resonate with current needs for secure but open information sharing.

Key motivating insights emerged from reflecting on the described challenges and on the pursued goals:

- Considering decentralization is essential given that data is distributed among autonomous agencies.

- Differential access requires sophisticated fine-grained authentication, authorization, and security models compliant with specific privacy policies.

- Services ought to be tightly coupled with organizational policies to prevent privacy violations.
- Security needs to be an inherent property, not bolted-on after the fact.
- Given potential unknown uses of a service, the burden of service customization needs to lie on the user and not on the provider.

The context of our work is decentralized information systems, whose individual constituent services operate under multiple authorities. Our ***hypothesis*** is that a software system rooted in dynamic architectural principles involving capability-based security and computation exchange, along with formal policy specifications can offer policy-based differential access to personal information services, while simultaneously enabling user-controlled service customization in decentralized contexts.

Along with the described insights, our hypothesis led to the formulation of ***our approach*** towards service customization and policy-driven differential access to personal data and our evaluation methodology (detailed in section which involves:

(a) the combination of formal policies and architectural principles and techniques—grounded on mobile code and capability URLs—for achieving fine-grained customization and policy-based differential access to personal information and computational services (detailed in sections 6.1 and 6.2);

(b) the assessment of the suitability and practical feasibility of the proposed techniques through conceptual and technical analysis, and experimental application development (section 7), and;

(c) the evaluation of our approach in the context of the healthcare domain, specifically addressing electronic health record management systems, based on comparative analyses and on patient information access, sharing, and, use scenarios (described in section 8).

The ***contributions and outcomes of our research*** are:

(a) Setting forth novel techniques that, unlike current approaches and technologies, simultaneously enable providers to securely offer per-user, customizable personal information services that comply with the organization's data disclosure policies. In concrete, these techniques enable binding privacy and operational policies to an organization's capabilities. According to these bindings, consumer-specific services are dynamically generated, coherent with formally defined policies. We leverage formal policy languages in order to specify domain specific policies.

(b) The secure, privacy-aware, customizable use and sharing of personal information in decentralized, trust-dependent domains though architectural principles on computational exchange. We leverage the principles of the COAST architectural style, which are rooted on the decentralized, dynamic, and asynchronous bilateral exchange of, not only information, but computations among peers. We provide an architecturally sound reference application which simultaneously enables provider controlled policy-based differential access to personal data services and user-controlled service customization.

(c) Design guidance for using the developed technique by way of the COASTmed prototype (section 7). This system allows to (1) formally specify policies with respect of the access to patients health records; (2) bind these policies to specific system capabilities, and; (3) permit users to achieve the desired customization given their individual authority to access these services. Customization is achieved through functional composition, namely by combining the capabilities and data made available to them by the system.

(d) Insight into the advantages, limitations, and the appropriate application contexts of the developed techniques compared to other approaches with similar goals. We gain these insights through theoretical and practical analyses by way of system development experience, and comparative and scenario based evaluations.

Chapter 2

Motivation

Many domains share the described challenges regarding the secure access, integration, sharing, and diverse use of personal data distributed across agencies, where trust relationships between people and organizations are disparate, and so is the authority to access this data. We provide a few examples:

- *Financial management.* Participants in financial markets access and manage different aspects of individuals' finances. Banks maintain accounts and refinance loans and mortgages; credit companies track acquisitions and manage credit card balances; financial advisors manage our investments, bonds, and assets; the IRS keeps account of our taxes; stock traders access aggregate purchase trends; credit bureaus craft personal credit reports, and so forth.

- *Judicial system.* Lawyers, courts, law enforcement officers, investigators, penitentiaries, and defendants participate to share, obtain, disclose, and withhold data relevant to a case for their individual goals. Due to the competitive forces inherent in a trial, there are many rules and restrictions on the authority to access and use data relevant to the defendant, and the extent of such access.

- *Employment records* are a recollection of a person's work history. This data is of interest to many parties such as new employers, credit institutions, recruiters, and the government for assessing experience in a given field, as an indicator of stability and reliability, or for other types of background checks. However, not everyone is entitled to obtain all or part of this personal record; access is context dependent and requires the data owner's authorization.

- *Government agencies.* Diverse government branches hold distinct information about individuals. For example, the DMV records our authority to drive, the IRS holds our income and tax data, the Social Security Administration manages our retirement earnings, and so on. Given the autonomy of these institutions, sharing information among them for diverse reasons—such as gathering more information about an individual or aggregating data for statistical studies of the population—is cumbersome, even when authorized. To somewhat deal with this complexity, digital government applications are aimed to improve communication between citizens and their governments, however mired with legal and technical challenges to preserve individuals' privacy [222].

- *Healthcare.* The healthcare industry involves a network of users and uses of health data for various (often conflicting) purposes. Effectively accessing and sharing medical records among healthcare providers has been a long standing challenge—time-consuming and bureaucratic processes delay the delivery of healthcare. The sensitive nature of this data requires rigorous disclosure control based on the data addressee, as well as on the context and purpose of use.

"Collaborative systems require making information accessible to all who need it [however] protect[ing] sensitive and confidential information, allowing only authorized personnel to retrieve and manipulate them" [275]. However, current web services, customization, and access control technologies are insufficient to effectively and simultaneously support fine-grained service customization and differential access to services. They either focus on service provision

or on authorization and secrecy without offering a integral approach for supporting both service consumers and providers with respect to their (often opposing) needs and concerns. In section 4, an overview of these areas is provided, as well as discuss why these technologies do not properly address the described challenges when appropriate.

2.1 Motivating Domain: EHR Management

The healthcare industry involves a network of users and uses of patient data for various (often conflicting) purposes. The problem we approach is typical in this domain, where the difficulty of securely sharing medical records among various stakeholders—patients, doctors, researchers, insurance companies, and government agencies—is a well-known problem [67]. Healthcare is mired in bureaucratic processes; paper-based medical records are still used, sharing is frequently done through faxed documents upon patients' requests, and patients are repeatedly providing the same information, even to the same healthcare provider. Effectively accessing and sharing health records among patients, healthcare providers, and other involved parties has been a long standing challenge. In addition, trust among the stakeholders in this domain is not homogeneous, but disparate trust relationships are held, and so are the types of services and data exchanged, the purposes for exchange, and the restrictions imposed for accessing the data.

To ameliorate these time-consuming and often bureaucratic processes, the healthcare domain is increasingly seeking digital optimizing technologies. For example, the adoption of electronic health records (EHR)—the recollection of data related to a patient's medical condition and treatment across time in a digital format—is on the rise. EHR management systems include the technology to manage—add, delete, update, organize, and share—digital records. Recorded information is, for example, patient identifying information, symptoms and complaints, clinical observations, laboratory tests, diagnostic imaging reports, treatments, ther-

apies, drugs administered, allergies, legal permissions, and the like [97]. Some EHR systems additionally have tools for scheduling appointments, requesting referrals and prescription refills, and decision support tools such as drug interaction finder [90]. Researchers and system developers make a distinction between EHR and personal health record (PHR) systems; whereas the former serves health care professionals, PHR systems are designed for end consumers—i.e. patients—so they can manage their own health information and be actively involved in their own health care [251]. PHR examples are Microsoft HealthVault and GoogleHealth [184].

Despite the availability of open source and commercial EHR and PHR systems and the perceived benefits of their use—improved quality, safety, and efficiency—adoption is slow and few healthcare practices (barring large medical groups and hospitals) have fully functional EHR systems; cost, unsuitability of tools to practice-specific processes, uncertainty of returns on the investment, the transience of vendors, privacy and confidentiality concerns, and the resistance natural to breaking away from traditional processes are some of the most significant barriers of adoption [35][90]. Although an Integrated Care EHR is meant to be a "repository of information regarding the health of a subject of care in computer processable form, stored and transmitted securely, and accessible by multiple authorized users" [141], current technology still falls short in living up to this definition; patients and providers still find themselves navigating in a complex healthcare system and in the need of immediate access to their own health records. What remains constant with regards to the collection and use of health data—and more generally of any private data despite of the domain—is the need for "selective information sharing" [277] among multiple decentralized parties.

Therefore, choosing the healthcare domain as a source of our examples, practical experiments (section 7), and evaluation scenarios (section 8.1) to approach the described research problem is deliberate:

- Despite the advances made in information system technologies, sharing medical data involves a chain of time-consuming processes mostly involving traditional methods of communication (e.g., phone, fax, mail).

- Patients often lack immediate access to their health information or to healthcare associated expenses.

- The sensitive nature of the information, legal strictures, and privacy and security threats related to the inter-organizational exchange of health data require rigorous and secure authorization practices and fine-grained dissemination control (based on user, context, and purpose of data use) to protect patient privacy.

- Patient information disclosure is contingent on a complex set of regulations—patient privacy is protected by the Health Insurance Portability and Accountability Act (HIPPA) (albeit fraught with ambiguous semantics) as well as by organization-specific privacy policies. Health information systems are created without tight conformance to these rules, leading to information authority abuse and privacy breaches in this field [69][75].

- It is not clear how healthcare processes, systems, and services capture or are bound to privacy policies and regulations.

- It is extremely important to effectively document, communicate, and perform evaluations on diagnoses, procedure outcomes, and therapies—the need for distributed and decentralized health information systems is imperative [49].

- There is an essential need for the effective and safe exchange distributed health data both for treating patients and to support socially beneficial privacy-preserving data mining and analysis practices for domains such as public health [159][73][162].

- Over the last decade, electronic health record systems have been increasingly adopted and have become core applications within hospitals [134][49]. These systems not only engender social benefits, but reduce costs and optimize processes.

- The Government is actively promoting and funding through the Health Information Technology for Economic and Clinical Health Act (HITECH) (issued by the United States Department of Health and Human Services in 2009) the adoption and "meaningful use" of EHR technologies; this is a good opportunity for the technological flourishing of the healthcare domain and the market growth for EHR systems [52].

- Agile and secure access and manipulation of personal data can arguably facilitate more effective healthcare delivery [90]. In addition, healthcare systems "improve the quality, completeness, depth, and accessibility of health information" and enable communication among parties involved in healthcare [91].

2.1.1 Motivating Scenarios: Obtaining EHR Data

The following scenarios address shortcomings in the domain regarding the lack of immediately available data, systems and services policy compliance, patient information privacy and security, the invariability and inflexibility of information services, and the integration of distributed and independently managed data.

- A patient visits a hospital for the treatment of some medical condition. As part of the treatment process, the patient's demographic data as well as data related to his symptoms and diagnosis is collected and entered into the hospital's electronic health records database. At a later date, the patient requires access to his medical record held by this hospital. In order to obtain a copy of his records, he needs to make a formal request either by mail, fax, or in person.

- A patient makes an appointment with a diabetes specialist. The doctor requires blood tests results performed at a laboratory a week prior to the patient's visit. The patient calls the lab to requests that test results be sent to the physician. A few days later, the complete lab tests arrive by mail at the doctor's office (figure 2.1).

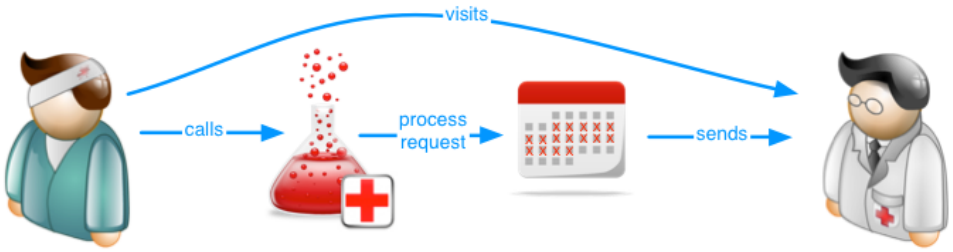

Figure 2.1: A patient in the effort to access and share his EHR.

- Academic researchers are investigating the relationship between diabetes and cardiovascular diseases. For this purpose, they require access to health records from multiple sources. After numerous conversations with healthcare providers and depending on the hospitals' administrative workloads and the time it usually takes to process this type of requests, they obtain various sets of anonymized records, information released after careful analysis of internal privacy policies. They extract and standardize the data (given in several different formats, discarding what is unneeded) and run a specialized analysis algorithm on this data (figure 2.2).

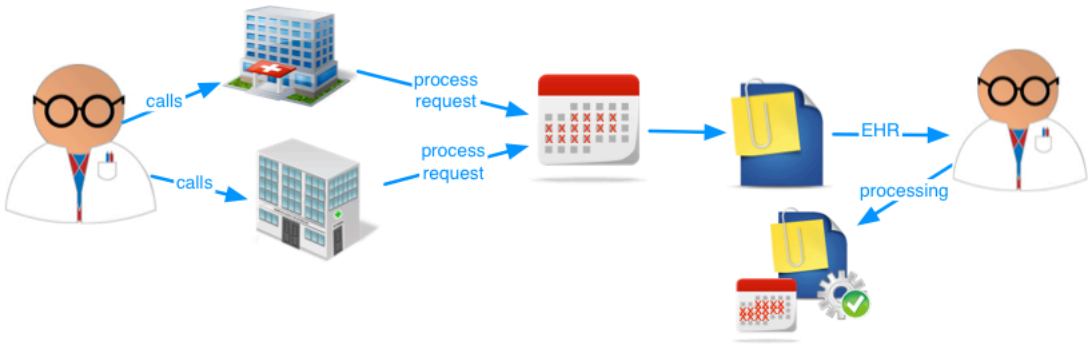

Figure 2.2: Researchers trying to obtain and organized distributed medical information.

- A patient diagnosed with hip osteoarthritis requires a hip replacement. Concerned about the procedure's cost, the patient calls her doctor's office, the hospital, and her insurance company regarding costs and coverage. Due to complex healthcare and insurance processes, the patient receives incomplete information. After the procedure,

the providers bill the insurance and several weeks later the patient receives an invoice for an amount well in excess of the anticipated costs (figure 2.3).

Figure 2.3: Patient trying to obtain data from multiple, inter-related service providers.

Thus, obtaining the required information to which each user is entitled to access is time consuming and binds users to bureaucratic processes.

Chapter 3

Research Questions and Goals

The challenge of decentralized, secure, and meaningful personal information sharing is common to many domains. Cumbersome (often manual) processes and the shortcomings of current technologies in appropriately addressing these problems lead us to ask the following overarching research question: *can we enable the different participating parties in a given domain to access the information and computation capabilities they need, in the way and at the time required, and within the boundaries imposed by the service provider by law or for the sake of privacy?* From this main question we derive more specific questions, namely:

1. How can providers of personal information—these being individuals whom the data describes or organizations which collect and store this information—safely expose this data according to specific trust relationships?

2. Can consumers in turn customize these services, manipulating personal data to fit specific information needs, and integrate remote data provided by those decentralized services? If so, what are the challenges and requirements for customizing such services?

3. What are the principal design decisions for a software system to simultaneously enable customization and policy-based differential service access?

4. Can such a system be developed in practice and what are the appropriate technologies for doing so?

Inspired by these research questions, our research has two overarching goals. First, enable differential access to a provider's services—both data and computational capability—where the provider discloses information according to privacy and operational policies. Differential access permits providers to distinguish among users, treat users idiosyncratically, maintaining multiple trust relationships, restricting and expanding the per-user availability of information and functionality.

Second, enable service customization, allowing service consumers to manipulate personal information and use the services' computational capabilities at will, however within the given authority and rights to access these assets

Concrete objectives that these overarching goals encompass include:

1. *Enabling fine-grained control over the access to a provider's services (both data and computation capabilities) by supporting differential access according to specific trust and legal relationships.* For example, according to a hospital's privacy policies, a patient is entitled complete access to his health records, but insurance companies may only access patients' performed procedures' dates and codes.

2. *Allowing a service provider to revoke the capabilities granted to service consumers according to changing trust relationships.* For instance, a hospital can deny future access to de-identified medical records to a researcher whose access agreement has expired.

3. *Enabling service composition and customization that allows the user to fulfill specific needs.* For example, a researcher can run a sophisticated analysis algorithm for type 2 diabetes on a population's lab results.

4. *Enabling integration of information from different sources under different spans of*

authority. For example, the CDC may use services from various hospitals to monitor and detect epidemic outbreaks.

The proposed approach described in subsequent sections are directed towards answering to these research questions and fulfilling our goals.

Chapter 4

The Context of this Work

Several communities have addressed decentralization, data management, service provision and composition, customization, authorization and access control, trust, privacy, and information integration standards, all relevant topics within our research. We highlight some of this work in light of the challenges we attempt to solve, providing the context of our research.

4.1 Software Architecture

Since the mid-nineties, many academic papers and books on Software Architecture have been written, refining the field's concepts and addressing the overlapping and iterative stages of software development—from requirement elicitation, to design, and to implementation and testing [213][235][34][252]. The architecture of a software system involves the overarching design decisions governing its behavior [252]. There are however other meanings of software architecture. For example, Perry et al. refer to architecture as a set of elements (processing, connecting, or data), form, and rationale; form determines the relationship among elements and rationale involves the motivation of design choices for satisfying system constraints

[213]. Others consider architecture as as a collection of interacting components [116]. More recent studies view architecture as an arrangement or configuration of components—which encapsulate data and functionality—and connectors—which enable the interaction among components [34][252]. In depth studies, for instance, address software connectors and their application context, such as Mehta et al.'s connector taxonomy [189].

Much research has been made towards capturing software architectures, namely on methods to model software as a set of components and connectors. Designing architectures is about abstracting the fundamental properties of the system from the implementation details, and therefore deal with the complexity and intangibility of software in a tractable way. Modeling methods range from graphical boxes and arrows diagrams such as those found in UML notations [124] to more formal representations such as those described in architecture description languages (ADL) [188]. A variety of tools address these methodologies, supporting the modeling process. For example, Rational Rose models UML diagrams [218] and Archstudio [88] models architectures with xADL, an XML-based ADL. Other tools support the process of collaborative design, for example Calico [182] and other e-whiteboard applications [66][85].

An important topic within Software Architecture are architectural styles—named collections of design decisions that constraint software design in specific ways, are appropriate for a particular development context, and which elicit desired properties in software systems [252]. Examples of architectural styles are pipe-and-filter, layered, object-oriented, blackboard, client-server, and event-based just to name a few [252]. Our research deeply relates to the notion of architectural styles, since our approach is grounded on the COAST style, described in following sections (section 5.1).

Lastly, design patterns sit closer to software implementations, providing solutions for recurring design and development problems. For example, the *observer* pattern where an object notifies a list of observer objects regarding changes in state is widely used within event-based systems. Gamma et al. provide a comprehensive account of design patterns [114].

4.2 Decentralized Systems

Decentralized systems are composed of distributed components managed by independent authorities. Extensive work has been devoted to communication protocols, availability, and fault tolerance in distributed and decentralized systems. Decentralization has also been discussed within autonomous systems, decision and control, signal processing, real-time systems, security, data management, networking, e-commerce, among others. E-commerce systems rely on the engagement of multiple parties; agents automate product and merchant brokering, product buying and selling, and negotiation in electronic markets [126]. Decentralization also challenges collaboration between geographically dispersed software development teams where the tradeoff is between collaboration and autonomy [39]. Decentralization is at the core of P2P computing [191], file sharing [233] and decentralized multicasting [63], where peers collaborate to complete tasks and share data. Decentralization has also been considered in the context of system availability, re-deployment, and runtime reconfiguration when a proprietary system is part of a larger network of cooperating systems [180]. Middleware [98] is an extensive field not only dealing with interoperability and synchronicity, but as well with decentralization given the communication and coordination difficulties between autonomous processes. However, the prime example of decentralization is the Web, which has scaled to be the largest decentralized system driven by a set of architectural principles [107]. The common trait of the described research (including ours) is communication, collaboration, and coordination between various entities for diverse purposes, ours dealing specifically with the decentralized access, sharing, and management of personal information.

4.3 Information Management Systems

This field has greatly evolved from centralized relational databases to dataspaces of diverse, distributed, and interrelated sources of free text, structured information, and semistructured data [110][220]. The Web itself grew from a distributed, hypermedia-based information management platform [42]. The challenge still is to integrate information fragmented by location, device, application, and ownership, challenges for which researchers have proposed diverse management strategies such as grouping, semantic tagging, linking, and relations-based approaches [152]. In our context, the challenge is to create privacy-aware services for individuals and organizations to manage their information across autonomous, distributed, and disparate sources. This challenge is related to personal information management [145] given that individuals may, to some extent, share control and ownership of their data with many organizations. For example, in the healthcare domain, accessing information on-the-spot from hospitals, physicians, and labs can optimize the delivery of healthcare [216]. Sustainable information systems ought to be (rather dauntingly) open, scalable, flexible, portable, distributed, standard-conforming, semantically interoperable, service-oriented, user-accepted, applicable to any media, and lawful [51]. Substantial research is focused on semantic search and interoperability [50][172][220], given the diversity of terminologies organizations adopt, as well as on system scalability and distribution [271]. The information management and security needs in decentralized contexts such as healthcare remain substantial.

4.4 Web Services

4.4.1 Service Provision

Service Oriented Architectures (SOA) [100] have been proposed to promote integration and interoperability by enabling software systems to access information and applications exposed by organizations through web services. Competitive technologies to implement SOAs include SOAP-based [16] and RESTful web services [223]. SOAP-based services rely on a triad of technologies: UDDI repositories to register and search for services, WSDL specifications to describe services in terms of inputs and outputs, and XML-based SOAP messages which provide a standard format of communication. Extending these technologies, the WS-* specifications provide a wide variety of additional services such as security, reliable message delivery, and service coordination. In contrast, RESTful web services follow the constraints of the REST architectural style [107] to access and manipulate resources through standard HTTP operations.

A significant drawback of these technologies is that providers are solely in control of the offered services and it is the burden of the user to manipulate syntactically and semantically the obtained information to fit specific needs. Current web services do not differentiate among users, but are mostly publicly offered through rigid, static, and uniform interfaces. As a workaround, private UDDI directories have been used to provide access to internal services [83]. There are, however, significant problems with this approach. First, it entails a very coarse-grained access policy; service is granted to anyone who has access to the directory. Second, there is no mechanism to prevent access to this service if unauthorized parties somehow obtain access to the private UDDI. Third, UDDI is an standard that failed to be widely adopted; public UDDI registries such as those offered by IBM and SAP have gone out of service. An alternative for offering private web services (discussed in developers forums) is to run these services within private firewall-protected networks, allowing only in-house

access or through a VPN. However, these are awkward workarounds and hacks which, alike to the private UDDI solution, only provide a binary access control policy. SOAP authorization headers provided with user and password information have been also used to access services. While this approach allows treating users individually, there is a great overhead to provide differential services in such way. For decentralized SOAs, authentication, secrecy, and integrity are necessary but insufficient for asset protection and service customization.

4.4.2 Service Composition

Developers solve more complex problems through service composition, namely combining services to solve specific problems and building applications such as mashups—systems composed out of third party services [274]. Popular service composition approaches include BPEL, OWL-S, DAML, Web Components, XLANG, WSFL, WSCL, Petri Nets, among others [19][190]. While some of these methods are implementation-oriented (e.g., BPEL and OWL-S), others are for the purpose of specification and analysis (e.g., Petri Nets and FSM). These languages allow composing services through the orchestration of service calls, therefore explicitly describing the sequence in which services are invoked and the inputs and outputs that link these services. Some of these languages are oriented to analysis and verification, but it is not straightforward how to proceed to an actual implementation. Additionally, there are specialized development environments such as SWORD [215] and Self-Serv [40] to support the automated and semi-automated composition of services.

The shortcoming of some service composition technologies is having to use several, often difficult, XML-based languages and standards to achieve basic compositions, the difficulty of describing complex services, and resulting bulky and unmanageable service descriptions. Current web services constrain developers to "embrace many technologies that must be integrated in an intricate manner" [209]. Also, in contrast to our approach (section 6), it is

not possible to compose complex services or "flows" from atomic message exchanges [240].

Moreover, recall that the building blocks of these service compositions are rigid, static components which do not provide the flexibility of customization. Therefore, it is difficult to find the required services to achieve the desired compositions, unless all services are developed purposefully to work together, which is not the case in decentralized settings as the ones we portray. Even then, the resulting service may not precisely fit the needs of all users.

However, the most significant drawback of these technologies is that they are mostly orchestration languages, where only one party controls the business process interactions [209], an unacceptable option for our problem domain whose essential characteristic is decentralization. This is for example the case of BPEL4WS, one of the most widely used service composition technologies. In BPEL4WS, executable processes, similar to a flow-chart, describe the order of activities, the parties involved, the messages exchanged, and exception handling rules [264]. For example, in a travel scenario the travel agent solely controls the interactions with a customer, dictating a fixed sequence of steps that the behavior that both a customer and the provider need to perform in order to schedule a trip [260].

In service choreography, independent parties observe the rules for interaction for standalone message exchanges. The most popular choreography language in current web service technologies is the Web Services Choreography Description Language (WS-CDL), which specifies contracts involving the common observable behavior of all participants [209]. "WS-CDL is a language for specifying peer-to-peer protocols where each party wishes to remain autonomous and in which no party is master over any other" [261]. So for example, a choreography description may specify the publicly observable interactions that parties in the roles of buyers and sellers may have. However, WS-CDL lacks explicit support for multi-party interactions—it does not compose nor execute multi-party processes. WS-CDL is a design and not an implementation artifact—its descriptions are used to generate web services code skeletons or internal BPEL processes to assist with interoperability [32]. So in the buyer and

seller example, the WS-CDL may be used to generate a buyer-side BPEL which describes what are the sequence of seller's service invocations, and therefore the information exchange from a single party's point of view. Therefore WS-CDL is not standalone, but requires integration with WSDL (to bind to specific WSDL interfaces), BPEL, and other web service standards; WS-CDL is far from a simple service composition solution.

4.4.3 Service Customization

Researchers distinguish between customization and configuration, the former requiring source code modification or extension [247], while configurable services explicitly specify the possible variations. For example, in [245] a meta-model specifies the variability of a service so that the set of possible configurations can serve different users; domain experts specify pre-configurations for different types of users, and users can create service variants.

Most customization approaches leverage semantic web technologies to: transform services configurations published in directories to obtain others more suited to specific needs [228]; automatically select services that meet user preferences and constraints through semantic translation [181]; use ontologies to transform customer requirements into high-level service process models that are implemented by services discovered at run-time [61]; recommend services by performing a similarity evaluation between a user's parameters and service offerings by measuring the distance between concepts in an ontology [59]. Other rule- and policy-based approaches translate user preferences into rules in a high-level policy language [138] and allow users to submit customization requests to service providers in order to build new services based on the providers' policies [171]—this approach requires the provider to actually fulfill those requests.

Orchestration languages also allow service users to configure services through a sequence of service calls. Indeed, composition is closely related to customization, given that composition

allows customizing a service. For instance, Srivastava et al. refer to customization when both RDF/DAML-S and BPEL4WS allow selecting at runtime among different branches of execution constituted by different service compositions [240]. As will be seen in subsequent chapters, we achieve in our approach service customization through service composition, thus exploring this relationship. In the aforementioned typology, our approach is closer to customization, therefore pre-defined configurations controlled by the provider do not exist, but offered services can be used as building blocks to compose custom services.

Despite the availability of the described approaches, little attention has been paid to service customization. Moreover, these techniques provide very limited customization capabilities, by and large pre-defined by the service provider, where users do not have much ability to customize services to their own needs. In addition, these approaches mostly rely on the traditional web services stack, hence bear all the disadvantages previously discussed.

4.5 Service Security and Access Control

Providers may want to offer their services only to some users, defining per-user authority over those services. To that end, authentication protocols verify the identity of parties [199]. For example, third party two-way authentication services such as Kerberos mutually authenticate communicating parties [243]. These protocols are based on public-key and other encryption algorithms such as password-based one-way encryption functions [164]. In addition, digital signatures demonstrate the authenticity of a message which cannot be altered or repudiated [199]. With these protocols, systems can enable name lookup, principals grouping, program loading, delegation, access control, revocation, and identity management [165].

Given an authenticated identity, access control mechanisms authorize the access to assets (e.g., functions, databases, devices), bestow the corresponding permissions, and prevent

using assets in unauthorized ways [175]. Authorization describes what each user is allowed or not to do. For example, the model described by Lampson identifies a set of principals, objects (e.g., files), requests to operate on objects, and a reference monitor which grants or denies a request [165]. Traditional access control models include: (a) discretionary access to individual objects based on user identity and on user-specific rules; (b) mandatory access is based on subjects and objects categories, and; (c) in role-based permissions are associated to roles [231]. In addition, domain-specific identity management and access control mechanisms, for example, in the context of healthcare can be found in Chen et al. [68] and [15]. However, it has been argued that none of the traditional access control models on its own is sufficient in complex, large-scale decentralized environments such as federated healthcare [15].

Access control and authorization models have been proposed for SOAs, such as models based on principals and objects' attributes [275] and workflow-based models (i.e. according to workflow processes and policies) [139]. Some web service providers also rely on network level security mechanisms such as HTTPS and digital certificates for data transmission and authentication [204]. The disadvantage of many of these models (e.g., Kerberos) is the centralized authentication, key escrow, and dependency on certificate authorities.

There are software systems for which security—authentication and authorization—is core to the application (e.g., operating systems). But although these security models and protocols are widely adopted, well-founded technologies, they are often tacked on software systems without being central to the architecture. Therefore, it depends on system developers to wisely choose the appropriate technologies so that the implemented system has no security gaps. As we describe in section 6, authorization and secrecy are a fundamental part of the architectural principles of our approach, thus security is a given, there and always, and not a mere afterthought.

The SOA world has also established security standards as part of their WS* stack. WS-Security [195], for example, includes digital signatures, encrypted messages, or identifying to-

kens in SOAP messages' headers [44]. Other associated specifications and extensions to WS-Security are: WS-SecurityPolicy specifies web services constraints, capabilities, and requirements in terms of policy assertions [242]; WS-Trust provides asserted credentials through the exchange and brokering of security tokens to establish trust [194]; WS-Federation enables groups in different authority spans to authenticate and authorize access to each other's resources [174]; WS-SecureConversation allows defining secure messaging semantics, contexts, and session keys [241]. Other OASIS standards are Security Assertions Markup Language (SAML) and eXtensible Access Control Markup Language (XACML) which allow the exchange of security-related data and specify role-based access control rules [204].

However, these XML-based standards are vulnerable, suffering, for example, message rewriting attacks [44]. We agree with O'Brien et al. [204] in that interoperability (and security) start to fail on services based on a complex specification stacks—other than WSDL and SOAP—when they are not adopted by all vendors or the same versions are supported. Also, developers often resort to non-standard workarounds such as service access through virtual private networks in order to provide (at least) binary authorization to services: users are either allowed or forbidden to access a service. WS* technologies do not enable fine-grained, per-user access, and selective use of web services, but are limited to adapt traditional access control technologies to SOAs.

4.6 Privacy

Privacy concerns have been addressed in many fields such as networks and communications, UI design, and HCI. Considerable amount of work has been done in the context of privacy and data mining practices. An array of privacy mechanisms have been proposed to prevent re-identification of individuals in anonymized records containing sensitive information. Privacy methods include data randomization, data swapping, and record distribution

among autonomous hosts [12][74][108]. More sophisticated privacy preserving methods are for example k-anonymity [249], l-diversity [178], t-closeness [168] and differential privacy [95], each one improving the others' deficiencies to minimize the risk of re-identification. A persistent challenge in this field, however out of our research scope, is the inference problem in the face of existing personal background information [102][198]—namely, when sensitive personal information can be inferred from other available data. A related line of work is that of privacy concerns when using personalized services. Wang proposed a framework for privacy-enhanced personalization that follows product line techniques to model and enforce user-defined privacy constraints in web personalization [262].

4.6.1 Privacy Concerns in the Healthcare Domain

One of the subjects individuals are most private about is their health information. Malicious or unintentional disclosure of a person's present or past medical condition can have serious social and economic consequences such as ostracism, insurance coverage denial, and medical identity theft, as well as other minor issues such as targeted marketing [184][17].

The Health Insurance Portability and Accountability Act of 1996 (HIPAA) was issued by the U.S. Department of Health and Human Services [96] with the goal of protecting patient health information. HIPPA provides the guidelines for the collection, use, and disclosure of health data, protecting "individually identifiable health information" (e.g., name, address, social security number). Entities covered under HIPPA are health care providers, health insurance agencies, and health care clearinghouses. The rule authorizes the digital transmission of health data for patients' treatment and health delivery operations, and payment of health services purposes, as well as the transmission of de-identified data for research and public health purposes. These agencies are required to disclose the minimum necessary protected health information needed for specific purposes [45].

Patient's privacy concerns and the potential risk of identification within de-identified databases have led to the research and development of various privacy-preserving methods and tools such as statistical privacy for databases [12], web search [270][272], location [161], and healthcare-specific [30]. In fact, data protection and privacy is one of the most fundamental requirements of EHR applications [137] demanding not only organizational but technical safeguards to comply with HIPAA [70].

Healthcare providers establish specific privacy policies regarding collection, use, and distribution of health data to comply with both HIPAA rules and organization-specific work ethics and operation. "A privacy policy statement normally contains specific purposes for which data can be used or disclosed" [45]. We distinguish between privacy notice and organizational policies. Polices within a privacy notice are those presented to the patient with the intention of informing how the collected data will be stored and handled. These policies are either presented in paper at the time of service or in digital format when using online services. The Platform for Privacy Preferences (P3P), for example, allows websites to provide machine readable descriptions of their privacy notices, making them accessible to users [81]. Organizational policies are instead those providing the guidelines for operation and employee behavior within an organization. These policies may be included in organization manuals, employee contracts (as a confidentiality agreement), or may be implicit and passed on in the day to day operation of the organization. Our work is rather concerned with organizational policies given our focus on enabling services that provide differential authorization and rights according to specific privacy policies. Organizational policies may be further categorized in internal and external policies. Internal policies dictate the behavior and the information access rights within the organization, therefore guiding the clinical staff—general and specialist physicians, nurses, interns, residents, and administrative staff—through the workflow of a highly hierarchical domain with diverse levels of permissions. External policies instead regard the disclosure of information to people and agencies outside the organization. A system's operation thus needs to perform according to organization-established policies so that

the offered services effectively comply with the privacy standards that the institution has promised its customers and with internally-imposed regulations. Our approach is designed to explicitly bind services to organizational privacy policies (section 6.1.3).

4.7 Privacy Policy Languages

Privacy policy languages describe, in a machine-readable format, policies to protect people's and organization's privacy and to comply with privacy laws. Although these are the main goals, policy languages have different purposes such as: informing users and publishing organizational privacy policies; capturing users' own privacy policies; describing and enforcing intra-organizational policies, and; supporting access control mechanisms [163]. Bonatti et al. differentiate among and compare languages by the type of policies they can express: role-assignment policies, access control policies, privacy policies, and obligation policies [55].

Syntactic differences among these languages are in part due to their deployment context. Most languages are based on XML, the lingua franca for interoperability in web-based communication. For example, P3P allows websites to describe their privacy policies in a machine-readable format that can be interpreted by agents installed in users' computers [80]. A P3P Preference Exchange Language (APPEL) instead allows users to specify their privacy policies so that they can be compared to websites' policies for incompatible attitudes towards data use and distribution [79].

Our work is more related to languages which capture obligations towards customers' data and internally enforce legal and self-imposed privacy policies. Examples are IBM's Enterprise Privacy Authorization Language (EPAL) [23] and OASIS's eXtensible Access Control Markup Language (XACML) [203]. In EPAL, policies are a set of rules involving parties, actions, purpose, conditions, and obligations. XACML instead provides a request/response

language where given a query, an action is permitted or denied according to a set of privacy rules. In addition, OASIS "XSPA profile" describes how to use XACML to exchange domain-specific privacy policies and promote interoperability among organizations in the healthcare domain [202]. Compared to P3P-like languages, these make a finer-grained distinction of users, purposes, and conditions, having as well policy refinement capabilities [27].

Policies have also been expressed in deontic logic based languages, therefore based on rights, prohibitions, and obligations. Rei, for example, is a logic-based policy language designed for pervasive applications where mobile devices dynamically access nearby services and devices [147]. Rei has also been used to augment semantic web languages such as OWL-S (used to describe services' capabilities) with privacy annotations, namely rules based on domain-specific ontologies [148]. Basin et al. propose a language based on first-order temporal logic to specify security policies [33]. Ponder is also a language designed for the specification of security policies, including authorization, filtering, refrain, and delegation policies [84]. However, Ponder is oriented towards access control to network, storage, and application management more than authority over private information.

Although languages themselves do not guarantee conformance to policies, they can be used along other mechanisms to enforce conformance as proposed in our approach 6. Yee et al. derive a set of requirements for systems to be able to comply with defined privacy policies and propose an architecture for service-oriented systems that protect consumer privacy [273]. The Platform for Enterprise Privacy Practices (E-P3P) is a more comprehensive approach (thus not only defines a language) which enables the enforcement of privacy promises made to customers [154]. However, E-P3P focuses more on inter-enterprise management of policies rather than on organization-specific policies to support privacy-preserving services. In alignment with the goals of our work, Karjoth et al. provide a model where organization-wide privacy policies are enforced throughout business operations to protect personal data from privacy violations [153]. Particularly relevant to our work is privacy in the context of

Electronic Health Records (EHRs). The Cassandra system [37] for instance, includes a language for expressing security policies which is role-based and which supports credential-based access control—appropriate in our context of decentralized systems.

4.8 Policies and Services

Core to the contribution of our work is offering services that comply with the organizational privacy and operational policies with regards to individuals' data disclosure. Other research work has tacked comparable goals. For example, Kagal et al. [147][148], in the context of the Semantic Web, leverage ontologies to model capabilities, security requirements, and privacy policies to match users and request to the appropriate Web services. Their OWL-S Matchmaker tool then verifies that services fulfill request requirements, and that users comply likewise to the service's policies. Kagal et al. use the Rei policy language to describe security and privacy annotations. Although their goals are akin to our work, namely integrating policies with capabilities, our technical and operational approach are fundamentally different. Our solution based on computational exchange seeks to simplify policy-based differential service access, while Kagal et al.'s implementation is based on the notably complex WS collection of languages and specifications. Also, services need to be created for each user/user type (as opposed to being generated on the fly according to policies), therefore it may suffer scalability problems. In addition, our research encompasses not only the authorized exchange of information, but the user-controlled service customization. Kagal et al. tackle an issue that we want to include in our future work, namely inspecting as well the services' output for privacy policy violations, highly dependent on how services are implemented. Kagal et al. also consider the user's policies for service delivery. However, since our approach empowers users to customize their own services, they can computationally manipulate services' output to meet their needs.

Rezgui et al. [222] present an object-oriented alternative that relies on the combination of digital privacy credentials, data filters, and mobile privacy preserving agents (implemented as Aglets [166]) in the context of Web services for e-Government. Privacy credentials define the service access scope a user is entitled to and data filters use those credentials to select the appropriate and authorized data from the provider's local assets. Mobile privacy have the goal of preserving privacy at remotes sites by returning from the service not only information, but an agent that provides a privacy-aware interface to this information within the service user. Our research, at this point, is not concerned with how the information will be used and disseminated by service users.

More traditional approaches include mandatory access control models for web services, where policies are formally specified on a temporal-logic-based language and are processed by a policy enforcement engine which generates platform-specific access control components that mediate access to Web services [237]. Attribute-based access control is another alternative to protect confidential information, however making it accessible to authorized personnel though richer service-level semantics [275]; incoming service access request messages are forwarded to a "Policy Decision Service" for approval or denial. However, the disadvantage of this approach is the repetitive assessment of access permissions. In contrast, our approach generates (after the appropriate credential presentations) a per-user service access reference (a Capability URL described in 5.1) based on the existing policies; from then on the user is free to access the service without further identity presentation or policy approval processes.

The Web Services Policy Language (WSPL) produces policy rules as a sequence of predicates; rules involve, for example, authentication, QoS, messaging security, and privacy associated to web services [18]. WSPL is similar to our approach in that service users must formulate requests that are acceptable to the provider to obtain the required service results. More importantly, WSPL supports some limited version of differential access. For example, a policy may specify how much each member type is required to pay for service use. WSPL

also has strong considerations for decentralization—autonomous intermediaries may handle, under defined policy conditions, confidential personal information exchanged between service and consumer. This work is however different from our approach in that it is focused on the characteristics of the messaging protocols between service user and consumer, and on obtaining "acceptable" service outputs in terms of criteria such as format, encryption, and certification. These requirements are, in general, domain independent ones, while the policies we address are domain-specific, therefore associated with an organization's operation. Also, incoming messages are inspected and compared with policies to either deny or accept a request. In COAST—the architectural style on which our approach is grounded—each service CURL is generated for a specific user based on the organizations' policies, so there is no need to repeatedly inspect incoming messages against all service's policies. Also, compared to Rei—the policy language we use—XACML (language from which WSPL is a subset from) specifications increasingly become verbose and complex.

The SOA community provides the WS-Privacy specification which describes services (and thus organizations') privacy policies to a web client [196]. It is implemented as the P3P specification previously described. Other similar work involves a negotiation approach where user agents negotiate with service providers on the amount of personal information to disclose to obtain service results [254]; privacy preferences are specified as domain specific ontologies in the DAML language. Although this work is concerned with organizational data disclosure policies, the focus is on informing users of such practices and allowing them to compare policies with their own preferences, but not necessarily on deriving services from policies or on constraining services such that policies are enforced. Chou et al. propose a two-leveled access control policy where policies from both the service user and the service provider are accounted for, as well as the service's own security attributes, when selecting composite services (sequences of service invocations) through a service filtering process (services that the user is not allowed to invoke are discarded) [71].

More tangentially, the WS-Policy specification [29] defines the security requirements and constraints for parties exchanging messages (requirements such as "all messages shall be encrypted using the AES algorithm"). WS-SecurityPolicy extends WS-Policy by describing service constraints and requirements to secure messages as policy assertions. These specifications are, however, unrelated to the goal of overseeing domain-specific data-disclosure. COAST's infrastructure provides secure messaging so that security is part of every COAST application, allowing the architect to focus on the application semantics. Bhatti et al. further integrates WS-policy with the XML-based X-GTRBAC access control policy specification language to enable more fine-grained access privileges to Web services [46]. In this authorization framework, policies attached to different service components are combined to output the appropriate access control policy for a given role. Examples are provided in the context of EHR systems.

Policy languages have also been used to describe the conditions under which web services are offered or requested to assist in the process of service negotiation and service selection based on functional and non-functional properties [11].

4.9 Trust and Reputation

Trust management goes beyond computer security to handle the dynamic relationships between parties and to exchange information across organizational boundaries. Most trust and reputation models rely on personal experience and third-party recommendations [226]. In peer-to-peer networks trust relationships are built over time through the interaction between nodes [170]. For example, in P2P file sharing, each peer has a unique global trust value based on the peer's upload history [151]. In the Web of Trust (WoT) [164] trust is propagated through a chain of nodes or aggregate user opinions [125][170]. Other types of trust models are statistical—requiring a number of parties to recommend an entity—and

hybrid which also considers the credibility of the recommenders [170]. WS-Trust allows web services to be accessed based on a token issued by an authenticating Security Token Service. Trust and reputation models can be integrated into the foundations of a software system through the constraints of appropriate architectural styles (e.g., the event-based PACE style [248]). In the context of our sample domain, interactive trust negotiation to access data is at the core of an EHR systems architecture [14].

4.10 Ontologies

An ontology is the organized knowledge structure that describes a domain, involving concepts, their relationships, and facts [65]. Associations between concepts may involve specialization (B is a subcategory, type, or class of A), containment (B is a part of A), roles (A's role is B), attributes (B is a property of A), and more complex time-based, state, location, and causal relations [244]. Ontologies are essential in knowledge-based and domain-specific systems, and equally important to enable shared understanding and concept reusability in inter-organizational communication.

Ontologies can be captured through informal natural language, semi-informal structured natural languages, semi-formal artificial languages, or formal proofs and theorems [256]. Visually, an ontology can be described as a directed graph where nodes are the concepts and edges describe the relationships among them. Ontology technologies include both languages and tools to describe and formalize ontologies. Examples of such technologies are the Knowledge Interchange Format [119], WebODE, Ontolingua, Protégé-2000, WebOnto, Chimaera, OntoSaurus, and Loom [77][236]. Most of these technologies were built as standalone tools, however, some produce OWL-, RDF,- or XML-based definitions and database schemas that can be used by other systems.

Ontologies may also be explicitly specified through database schemas or implicitly found within existing databases. WebODE [78] and SOR [176], for example, use relational databases to store and perform inferences over ontologies. The difference between an ontology and a database schema, a quite subtle one, is often misunderstood. Uschold argues that while an ontology is the structure of some domain, a schema is the structure of a database, so the purposes of these technologies are different; the focus of a database is the efficient data storage and retrieval, while the focus of ontologies is to provide humans and systems shared understandings [256]. Other work considers both data models or information modeling techniques, yet ontologies are favored when specifying domain knowledge requires "enriched" meaning [183]. We add to this discussion by pointing out the difference between storing an explicit ontology in a database and deriving an ontology through inferences made on the structure of a database. In the former case, for example, a three column table may store an ontology where the first column specifies the relationship between objects named in the second and third columns. In the later case, we can infer relationships between objects, for example, though the existence of foreign keys.

Given the similarity and natural compatibility between ontologies and schemas (both ways of organizing knowledge), tools have been developed to transform ontology specifications to database schemas [25], and inversely tools to automatically extract an ontology from a database schema (e.g., Li et al. obtain OWL ontologies from relational databases [167]). In the former case, the purpose is leveraging databases' scalability, security, transaction management, and optimized search. However, a disadvantage of storing ontologies in databases is the difficulty to make runtime modifications [25], while domains concepts and their relationships are always evolving.

Ontologies are very relevant to our work since the domains we consider are complex and highly dynamic—numerous interacting parties exchange various aspects of personal information. Therefore, this complexity demands a structured organization of knowledge and

shared semantics for meaningful inter-agency data exchange. In particular, domain vocabularies are essential for the specification of domain policies.

4.10.1 Healthcare Information Standards

Standards in the health industry have been developed to achieve interoperability among health care providers and inter-agency medical systems to record, exchange, and process patients' health information. Several organizations are involved in the making of these standards such as the European Committee for Standardization (CEN) [2], Health Level 7 (HL7), the International Organization for Standardization Technical Committee for Health informatics (ISO/TC 215), OpenGALEN, and the Integrating the Healthcare Enterprise (IHE) industry initiative [97][234].

Standards differ in goal and scope; some standards specify the content structure of medical information, others information communication and security protocols, and others provide a shared vocabulary among health care providers [232][97].

Content structure standards. Examples of EHR content structure standards for exchanging biomedical data are the openEHR archetype, HL7 Clinical Document Architecture (CDA), CEN EHRcom (or EN 13606), DICOM SR, and the Medical Markup Language [97][234]. For example, openEHR archetypes are structured information models—expressed in the Archetype Definition Language (ADL)—which describe computable, reusable, discrete medical concepts such as "blood pressure" [36]. In addition, some of these protocols allow including multimedia—pictures and videos. Also, some standards are specific for certain type of content—DICOM is for medical imagery, existing already a standard library of domain knowledge [1][232].

Communication and security protocols allow submission, location, query, and retrieval

of electronic health records. They may also provide security services such as message encryption and user authentication (often through the Transport Layer Security (TLS) protocol) [97]. Examples of such protocols are EHRcom, Web Access to DICOM Persistent Objects (WADO), DICOM Structured Reporting (DICOM SR), the Retrieve Information for Display (RID), and the IHE Cross-Enterprise Document Sharing (XDS) [97]. Content structures are exchanged, for example, via XML, ebXML, or EDI messages. WADO, for example, defines web interfaces to access DICOM content. Some standards may define both content structures and access protocols, such as EHRcom.

Shared vocabulary standards provide vocabularies and code systems that support the semantic exchange between healthcare professionals and systems. Examples are the Logical Observation Identifiers Names and Codes (LOINC)[5][186], the Systematized Nomenclature of Medicine (SNOMED) [239], RxNorm (defines clinical drug names), the World Health Organization's Anatomical Therapeutic Chemical (ATC) Classification System, the International Classification of Diseases (ICD), the Current Procedural Terminology (CPT), the International Classification of Primary Care (ICPC), and The GALEN Common Reference Mode [97][232][131]. LOINC, for example, allows identifying and reporting clinical and laboratory observations so that healthcare stakeholders—such as physicians, hospitals, laboratories, pharmaceutical manufacturers, government agencies, and the like—can not only share, but automatically process, organize, and analyze information. Integration among these standards is achieved through technologies such as the Unified Medical Language System (UMLS) which maintains mappings among vocabularies [131] and the ARTEMIS middleware which enables interoperability between systems using diverse vocabularies [56].

Content structure standards such as HL7 messages may embed this standard domain knowledge. For example, openEHR archetypes may contain ontological definitions authored by domain professionals or as defined in standards (such as LOINC or SNOMED), as well as bindings between local and external vocabularies [36][97]. Similarly, the HL7's Vocabulary

Technical Committee creates the domain terms used in CDA documents to disambiguate domain jargon [232].

Despite the existence of structural and communication standards, the problem of interoperability, share understanding, and widespread use of electronic health records and the described standards remains [131]. Hindering the pragmatic interoperability among healthcare institutions are multiple, competitor standards, slow standardization processes, a lack of balance between clinical domain expertise and software engineering efforts, and proprietary interests, among others [131]. Moreover, in spite of semantic translation efforts such as UMLS and ARTEMIS, redundancies, errors, and ambiguity are inevitable given the amount of clinical information involved. The need to create and manage shared, inter-institutional EHRs lingers [234].

Chapter 5

Foundational Technologies: COAST and Policy Languages

Our approach includes a conceptual, technical, and experimental analysis of the suitability of combining a set of architectural principles and techniques with formal policies for achieving fine-grained customization and policy-based differential access to personal data services. Accordingly, this work exploits and contributes to recent studies in Software Architecture, specifically to the COAST architectural style for the design of decentralized, adaptive, and secure systems [121][122].

With the goal of providing policy-based differential access to services, we exploit the Rei policy language for defining privacy and operational policies. Rei is chosen among other policy languages after thorough evaluations (section 6.1.2) due to its compact, well-defined, and expressive logic-based syntax. These technologies, in combination, are core components of our approach for achieving the solution to the identified problem.

Following, we present the architectural foundations of COAST (section 5.1.1), its supporting infrastructure (section 5.1.2), and its comparison to other styles and technologies 5.1.3.

Additionally, we provide an overview and evaluation of a set of selected policy languages, leading to Rei as the technology of choice for implementing our approach (section 5.2).

5.1 The COAST Architectural Style

We have chosen to begin our studies using COAST for three fundamental reasons:

- COAST has a strong emphasis on decentralization, a fundamental characteristic of domains such as healthcare and e-commerce. Decentralization is at the heart of the problem we approach, involving multiple autonomous parties, multiple uses of personal information, divergent trust relationships among them, and variable authority to access personal information.
- COAST is designed to provide secure on-demand services which are created and terminated independently. Its foundations in mobility and the ability to compose computations from available assets provides a customization power not achievable with traditional web services. Also, its strong focus on adaptation allows evolving, expanding, and creating new services, necessary to cope with changing trust relationships in these dynamic domains.
- Lastly, and importantly, in COAST security is everywhere, always, and built into the provided supporting infrastructure; the supported capability-based security is appropriate to enable differential access to a provider's services.

5.1.1 Architectural Foundations

An architectural style is a named collection of design decisions applicable to a particular development context, which constrains a system's design to elicit beneficial properties in

resulting software systems [252]. The COAST architectural style was conceived for decentralized contexts were parties come and go as they wish, and create, terminate, evolve, and use autonomous services for their own purposes. COAST's design decisions are rooted in a paradigm of computational exchange, where peers bilaterally exchange, not only information, but computations. In COAST, the application state evolves through the dynamic and asynchronous transfer of computations, allowing the application to scale and evolve in a decentralized way.

The COAST principles are as follows:

- All services are computations whose sole means of interaction is the asynchronous messaging of closures (functions plus their lexical-scope bindings), continuations (snapshots of execution state), and binding environments (maps of name/value pairs).
- All computations execute within the confines of some execution site $\langle E, B \rangle$ where E is an execution engine and B a binding environment.
- All computations are named by capability URLs (CURLs), unforgeable, cryptographic structures that convey the authority to communicate. Therefore, computation x may deliver a message (closure, continuation, or binding environment) to computation y if and only if x holds a CURL u_y of y.
- The interpretation of a message delivered to computation y via CURL u_y is u_y-dependent.

Security. Given that communication and application evolution in COAST is by way of computational exchange, and therefore exposed to the security risks of mobile code crossing organizations' boundaries, it is imperative that security mechanisms are inherent to the architecture and not a development afterthought. In COAST security is everywhere, always.

COAST is based on two security principles. First, the Principle of Least Authority (POLA) [227] dictates that: (a) the default situation is the lack of access; accessibility should be based on arguments on why authority should be granted, and; (b) users should be granted the least amount of privileges necessary to complete their task. In COAST, security comprises authority (assets granted access to) and rights (use rights of those assets). Second, COAST sustains capability-based security [53] which provides an unforgeable "key" conferring both authority and rights. "Holding a key implies the authority to send messages to [an] entity or to pass the key to a third party" [53]. Similarly in COAST, a key is reified by a capability URL and entities are computations. These principles confine the authority and rights of computations communicating with and accessing the services of another one.

Execution environments. An execution environment $\langle$ E, B$\rangle$ is a execution engine/binding environment pair that exists within some host. Multiple execution environments can exist within the same host. Execution engines are language-specific interpreters or compilers which execute code. *E* may enforce site-specific semantics such as limits on the consumption of processor cycles, memory, storage, or network bandwidth. Binding environments are a set of key/value pairs which provide the lexical context for executing computations, therefore bounding the functional capability of incoming computations. These environments may contain primitive values, data structures (including binding environments), and general purpose and/or domain-specific functions. In accordance with POLA, environments should contain only the necessary capabilities that an incoming authorized computation need to execute.

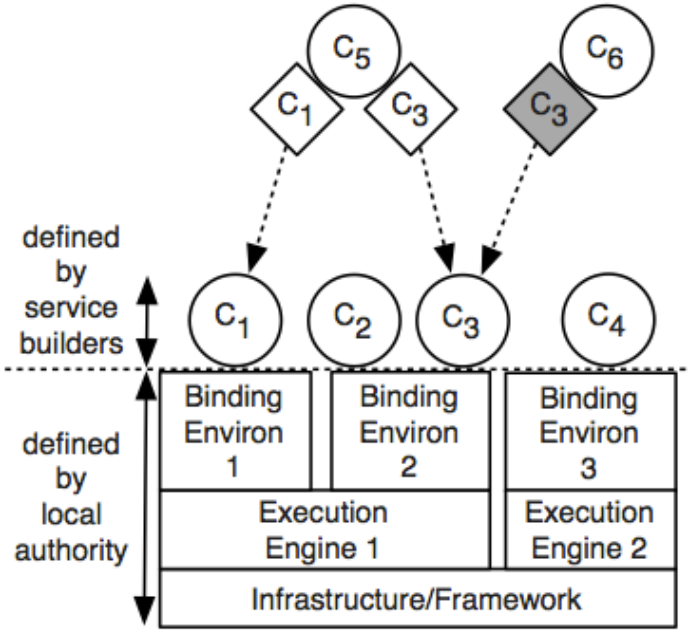

Figure 5.1: Notional structure of a COAST execution host [121].

COAST computations. A COAST computation is the execution of a closure c by execution engine E in the context of binding environment B. As described, B provides the local functional capability—the global bindings—available to that closure. c may specialize its capability by composing functions from bindings available in B. Also, closure c may gain additional capabilities by receiving additional binding environments in a message from trusted computations to augment B.

Capability URLs are the pivotal architectural elements providing capability-based security, conferring authority and rights to communicate with computations they name. A CURL u_x is a cryptographic, unforgeable, tamper-proof, and digitally signed (by x's execution host) reference to computation x which other computations use to send messages to x. u_x can hold additional metadata (including closures)—arbitrary semantic information understood and used by x to process messages sent using u_x. CURLs constrain the interaction computations can have with each other given that they are bound to specific binding environments. Therefore, a CURL u_x implicitly denotes the services offered by x, accordingly providing functional capability to closures sent in a message to x using u_x. CURLs contain all the information that the named computation will need in the future to serve the CURL holder(s). In addition, computations may gain additional capabilities by holding CURLs to other computations—computation x gains additional capabilities, those contained in computation y, by obtaining u_y. CURLs may cap capabilities as well, not only by addressing capability-limited computations, but through temporal and use count semantics.

Asynchronous messages. When a computation x sends an arbitrary message m to y using CURL u_y, both m and u_y are delivered to y. Messages may contain primitive values, data structures, binding environments, and closures. y needs to understand the semantics of m in order to evaluate it. A closure c, content of message m, leaves behind its global bindings at B_x. These are then "re-wired" with bindings at B_y. If some binding within m is not locally resolved, it is discarded. Otherwise, c is evaluated. If a "reply-to" CURL is provided, the

result is returned to the addressee.

Message interpretation is at the core of COAST's security principles; CURLs and their corresponding computations and binding environments prevent malicious code from being evaluated, since no local computation beyond the scope of B is possible. For instance, it is impossible for c to execute functions to access the file system at y since the required functions do not exist in B_y. Therefore, the execution site determines the capabilities and side-effects of all visiting computations.

Framed in Software Architecture concepts [252], COAST's *software components* are computations control system functionality and access to data. CURLs and asynchronous messages are *software connectors* which permit the interaction between decentralized computations. Figure 5.1 presents the notional structure of a COAST execution host (figure from [121]).

5.1.2 Motile/Island

Although architectural styles are detached from implementation details, some technologies are more appropriate than others in order to adhere to a particular architectural style. Some architectural frameworks and infrastructures have been developed to support building applications cohesive with an specific architectural style. For example, the Myx framework was purposefully built to develop conforming applications to the Myx style [87]. Similarly, the Motile/Island infrastructure [121] is provided to assist mapping COAST's architectural constraints to application implementations.

Motile is a single-assignment functional language deliberately created for the serialization and exchange of mobile code among COAST peers. In Motile, computation exchange is reified by mobile code. COAST computations (which include closures and binding environments), messages, and CURLs are implemented in Motile. In addition, to waive common

shared memory problems, all Motile data structures—used to build COAST primitives—are persistent and immutable [207].

Actors. COAST computations are implemented as *actors* [13], computational agents that are capable of receiving messages. In response to messages, actors either send a finite set of messages to other actor(s), execute some private computation, or *spawn* (create) one or more new actors. Actors are complex data structures that principally maintain a thread of execution by which messages are sent and received. Actors are initialized with specific binding environments, and therefore with an initial set of functional capabilities. Motile actors are named by one or more CURLs which provide specific authority and rights to other communicating actors.

Islands are the reification of COAST hosts—single address spaces described by an IP address/communication port combination. Islands host one or more actors. A *root* actor is a the top of an arrangement of *clans*—a semantic grouping of actors (e.g., image processing or audio playback clan). The rationale to this arrangement is modularity, separation of concerns, and decomposition of a COAST system into manageable units of computation. Communication between inter-island actors is secure and encrypted; islands are self-certified [149], associated to a public key, and issued CURLs are cryptographically signed. In contrast to centralized authentication approaches previously discussed (section 4), self-certification is independent of certificate authorities and in tune with decentralization.

Compilation and Serialization. Bedrock to the COAST infrastructure is a compiler which translates Motile closures into Racket, a dialect of the Scheme functional programing language. Closures are then serialized and transmitted to actors running on some other islands, where they are recompiled into Motile closures. Closures leave their global binding environments and are rebound with the available bindings at the actor receiving the closure.

COASTcast is an application for decentralized, dynamic, and collaborating HD video

streams which demonstrates how to build dynamic, decentralized applications with COAST (developed by UCI student Kyle Strasser). Computations embodying encoders, decoders, proxies, and pub/sub relays are dynamically created, manipulated, and reconfigured, moved from one host to another, and arranged to share SOA-style services to stream real-time video from and to fixed island assets (i.e. from cameras to displays). Computations hosted on different islands show how decentralized collaboration and inter-organizational processes can be carried out through computational exchange.

5.1.3 Comparison with other styles and technologies

COAST is a complex style which is composed by or shares properties with other styles, therefore obtaining as well their evoked benefits. For example, COAST is a peer-to-peer composite style, where autonomous peers communicate through a network; peers behave like either clients or servers within individual interactions. However, COAST also describes the security constraints of communication and the structure of peers. Also, alike event-based styles, parties in COAST communicate by way of asynchronous messages [117][252]. However in COAST, messages are not limited to events which parties broadcasts to one or more parties, but are also means of one-to-one communication and direct invocation.

COAST is also strongly founded on the mobile code style, where code is sent to be executed in a remote host [112][252]. Fuggetta et. al identify three types of mobile code: (a) remote evaluation (code is evaluated elsewhere), (b) code on demand (code is fetched from elsewhere and evaluated locally), and; (c) mobile agent (both code and data move elsewhere to execute, leveraging the host's resources). COAST style and mechanisms are able to implement these three paradigms which are application- and context- dependent. In comparison with other mobile code technologies, COAST and Motile/Island infrastructure is chosen over technologies such as Agent Tcl, Ara, Sumatra, Telescript [112]; COAST intentionally

supports weak mobility to protect local assets [122], while these technologies support strong mobility, therefore migrate both code and execution state, imposing additional security challenges. Other technologies supporting weak mobility are, for example, Facile, Aglets, M0, and Tacoma [112]. However, these technologies do not address potential communications constraints according to the individual user's privileges nor the conditions of code mobility; M0, for instance, executes the received code unconditionally. COAST explicitly imposes security constraints on the capability to communicate and execute code with remote hosts. In addition, COAST further addresses the context in which mobile code is evaluated. In other words, COAST is also concerned with the structure of components (i.e. hosts) and not only on the means for communication (connectors).

There are other styles addressing distribution and decentralization. For example, elements in the C2 style also maintain a message-based communication. However C2 relies on message routing intermediaries and it is strongly focused on achieving a distributed model-view-controller pattern, an important but distinct goal from ours [252]. Also, C2 enforces layered communication, in which event notifications and request flow unidirectionally from one component to the other. In COAST, any peer may be communication initiator or receiver.

In the distributed objects style, method calls to an object are performed across a network [252]. This style is itself a combination of the object-oriented and the client-server style. However, as analyzed by Taylor et al., interactions are mostly synchronous [252]. In addition, objects have uniform interfaces for all callers, therefore not appropriate for service customization. An ad hoc solution may be to customize services according to variation in method parameters, however distinction among users is not the focus of the style and this solution may not scale. COAST naturally enables combining services, while service composition with distributed objects would involve a series of method invocations to a single or multiple objects, locally combining individual results to perform subsequent invocations.

Representational State Transfer (REST), the style which describes the architecture of the

Web, is also appropriate for large-scale, distributed, and decentralized systems [107]. REST focuses on communication protocols and the structure of data elements, but is not concerned with the structure of components (clients and servers). However, the way web applications work has changed significantly since the academic paper on REST was published in 2002. Erenkrantz et al. recognize the dissonance between more recent web applications and the prescribed principles of REST [99]. For example, applications such as e-commerce rely on stateful interactions; cookies are used to resume previous interactions. They also found that others, such as web-based archives, require dynamic instead of static resource representations. Widely adopted SOAP-based web services are based on custom methods other than HTTP's uniform interfaces (however hijacking the GET method to perform non-idempotent operations) and are not amenable to intermediaries and caching [99][223]. Most relevant to our work, are emergent applications and technologies such as mashups and AJAX, which through asynchronous request obtain data and code to perform client-side computations [99]. The examples laid out by Erenkrantz et al. demonstrate that current needs require richer interactions that go beyond what distributed hyper-media (as described by REST) can offer.

5.2 Policy Specification Languages

With the goal of enabling fine-grained, trust-based control over a provider's services (G1), it is necessary to accurately capture trust and legal relationships among consumer and provider. Policy languages are appropriate for capturing service access conditions; formal policies are unambiguous and support avoiding conflicts given their proneness to automated interpretation and processing. In addition, the formal specification of policies facilitates their storage and categorization, promotes the use of standardized concepts, makes them more easily available and transmitted, and further enables interoperability between systems.

Accordingly, it is necessary, as a first step, to select an appropriate policy language for

capturing policies. As previously described (section 4.7), privacy policy languages have different goals. We are interested in leveraging a language that is able to describe intra-organizational policies (i.e. internal business rules), and therefore support access control processes to COAST-enabled services.

We evaluate a set of policy specification languages based on their expressiveness, hence their ability to represent selected organizational privacy policies. Our goal here is not to perform an exhaustive breadth- nor depth-wise evaluation of policy languages—thus neither covering all languages nor the extent of their expressive power—but to provide a representative set of alternatives to describe policies in our experimental domain. The analyzed policy languages are EPAL, XACML, Cassandra, PeerTrust, Ponder, and Rei. We briefly looked at languages such as Protune [54], SecPAL [38], RDM2000 [276], RT [169], PolicyMaker [48], KAoS [257], and WSPL [18]. However, these were not analyzed in depth either because the goals of the language were too divergent from ours (e.g., RDM2000 focus on role-based delegation), there was not sufficient guidance and example documentation (e.g., KAoS), or because we analyze a very similar language in semantics and syntax.

5.2.1 Evaluation Criteria

We evaluate the appropriateness of the following languages based on their purpose and semantics, and their ability to convey policies in our experimental domain—healthcare, and more specifically related to the access to electronic medical records. The language ought to be able to express subjects (people and organizations involved), their active roles, objects (to what access is being granted or denied), attributes, obligations, rights, prohibitions, conditions, exceptions, usage (purpose), and temporal constraints; a set of statements that include these semantics constitute authorization rules to access a service (i.e. data) [211].

In appendix A, we provide a collection of sample health care practitioners' privacy poli-

	covered concept	policy
P1	subject, object, right	Only cardiologists are allowed to access cardiac medical records.
P2	temporal constraint	No one shall have the ability to delete clinical information until the appropriate time period has expired (6 years).
P3	active role	A doctor should be granted the permissions assigned to the primary care physician of a patient (read and modify his/her EHR) only when the patient has designated him as the primary care physician.
P4	condition	A specialist physician (e.g., cardiologist) may prescribe drugs if the treated illnesses is related to his/her specialty (e.g., cardiac-related).
P5	usage	The hospital may limit the access to anonymous medical data only to authorized organizations for research purposes.
P6	prohibitions	Hospital staff other than physicians may not order lab tests.
P7	exceptions	A physician may order a lab test except when he is not part of the patient's care team.
P8	attributes, obligations	A person must be a custodial parent or legal guardian must provide proof of kinship to request access to the electronic health record of a patient under the age of 18.

Table 5.1: Privacy policies in the healthcare domain

cies to illustrate the type of authorization and rights parties have with respect to patients' data. These have been extracted from hospitals' and insurance companies' privacy policy documents (we resort to external policies given the unavailability of internal policies documents), as well as from relevant scenarios found in the academic literature and (if required) adjusted to be read from within a healthcare organization's—such as a hospital—perspective [255][205][277][17][187][210] [105]. The privacy policies we use for language evaluation have been selected from this collection (and modified as required to illustrate certain concepts), and are shown in table 5.1.

5.2.2 Languages Overview and Evaluation

Enterprise Privacy Authorization Language (EPAL)

EPAL is a language for specifying enterprise privacy policies involving authorization rights and data handling in information systems (W3C submission) [23]. In EPAL, a privacy policy is a list of rules; those that come first have precedence over subsequent ones. EPAL allows defining domain-specific elements with well-defined semantics to compose rules: ruling (e.g., allow), user category (e.g., sales department), action (e.g., store), data category (e.g., customer-record), purpose (e.g., order-processing), condition (e.g., the customer is older than 13 years of age), and obligation (e.g., delete the data 3 years from now). A rule must have a user category, an action, a data category, and a purpose.

An EPAL policy is a well-formed xml-based document (conforming to the EPAL schema) whose root is the `<epal-policy>` element. Policy documents include a domain vocabulary composed of one or more elements of type `<user-category>`, `<data-category>`, `<purpose>`, `<action>`, `<container>`, and `<obligation>`. For example, an electronic health record data category is formalized as:

```
<data-category id="EHR" parent="EHR">
  <short-description language="en">
    Electronic Health Record
  </short-description>
  <long-description language="en">
    Patient's digital record which contains the patient's health
   data
  </long-description>
</data-category>
```

Given this domain specific vocabulary reified as a set of declared elements, policies are

specified as in the following example: *email can be used for the book-of-month club only if consent has been given and age is more than 13* [22].

```
<rule id="rule1" ruling="allow">
  <data-user id="borderless-books"/>
  <data-category id="email"/>
  <purpose id="book-of-the-month-club"/>
  <action id="read"/>
  <condition id="consentToBookClub"/>
  <condition id="olderThan13"/>
  <obligation id="retention">
    <parameter id="days">5</parameter>
  </obligation>
</rule>
```

We provide the complete EPAL specification of the policies in table 5.1 in appendix B.1.

EPAL fails to successfully describe P2 which involves the time constraint "until the appropriate period has expired (6 years)". Although EPAL can express date comparison, it has no capabilities to naturally express date subtraction (e.g., `(current date - EHR modification date) >= 6 years)`. There are two workarounds to this problem. One involves keeping a counter as an attribute of EHR elements which keeps track of the days since the EHR was last modified. The second workaround—as suggested by the EPAL authors—is to keep a parameter `<years>6</years>`indicating that the data can be deleted after 6 years; the system requires understanding the domain-specific structure of the EPAL specification to interpret and provide the appropriate permissions according this parameter.

Although we described policies P4 and P6 in EPAL, the descriptions did not fit naturally with the semantics of the language. In the case of P4, we needed to associate a medical condition (e.g., arteriosclerosis) with a clinical speciality (e.g., cardiology). However making those kinds of associations is not straightforward. As a workaround we included an additional

attribute `TreatedConditionArea` to specify a matching speciality to a patient's illness.

P6—which expresses a prohibition for hospital staff other than physicians—had to be expressed in EPAL as an exception; first we wrote a policy that allowed physicians to order lab tests, followed by one one which denied hospital staff from ordering lab tests. EPAL mechanism to express exceptions is through rule precedence; when rules are in conflict, the first one is followed and subsequent rules are ignored. Rule precedence which requires the ordering of rules, however, can be difficult when an organization has hundreds of policies. P7, which is intended to express an exception ("a physician may order a lab test except when he is not part of the patient's care team"), was described more naturally as the rephrased condition "a physician may order a lab test if he is part of the patient's care team".

In EPAL (and in XACML as well), obligations have a different semantic from what we intend. An obligation is an activity a party needs to perform as required by law or as a condition for something else to happen. In EPAL and XACML, an obligation is an action that is taken as an additional step after some rule has been processed.

What is still unclear is EPAL's assumption with regards to default permissions and prohibitions. The question is whether any action is prohibited unless otherwise specified or if both authorizations and prohibitions need to be explicitly specified. For example, according to the policy "physicians can modify medical records", should a system prohibit EHR modification to everyone by default unless otherwise specified?

Lastly, EPAL is meant to express policies and practices which deal with the access to information (`<data-category>`). Policies such as "a parent or legal guardian must accompany a patient who is a minor" are not meant to be described in EPAL. Therefore, an organization is ought to use an alternative language for non-data-related policies and this is inconvenient at best. Ideally, a single language should suffice and be extensible to describe any kind of organizational and process-related policies.

XACML and the XSPA profile

Both XACML and XSPA are approved OASIS standards for secure and authorized communication and interoperation between organizations. XACML is an attribute-based language for defining access control policies and enabling the evaluation of access requests. For instance, requests to access a resource and the corresponding response are both described in XACML. The language model has three main components: policy set, policy, and rule. A rule specifies permissions or prohibitions, the effect or consequence of the rule evaluating to true, conditions, obligation expressions, and advice expressions. A policy is a set of rules. XACML policies are well-formed XML documents that conform to an approved schema.

XSPA provides a standard nomenclature for the exchange of healthcare-oriented security and privacy policies over XACML [202]. XSPA's purpose is enabling interoperation in the healthcare community through common understanding of specific permission codes that authorize various actions with respect to medical information. XSPA provides standard terminology implemented as element attributes such as subject id, provider id, role, permission, resource id, purpose of use, confidentiality code, allowed organizations to access a resource, and so on. XSPA leverages the HL7 Healthcare Permission Catalog [136] which provides a detailed and coded specification of permission types that can be used as values to the XSPA's `urn:oasis:names:tc:xspa:1.0:subject:hl7:permission` attribute. For example, the value `urn:oasis:names:tc:xspa:1.0:hl7:prd-003` stands for the "Review Medical History" permission.

Find the XACML specification of our sample policies from table 5.1 in appendix B.2. We only used XACML to specify the sample policies given that they involve more general specifications, while XSPA is more appropriate for more detailed permission descriptions. For example, compare the general "access to medical records" to more detailed ones such as "review patient identification" and "review patient medications", both which are implicitly

included in the more general permission.

Compared to EPAL, XACML is more expressive, however at the cost of increased complexity. XACML can describe who, what resource, what action, and when to access specific medical data. Also, XACML allows resolving the evaluation of conflicting rules. For example, it may be stated that a *deny* permission returned after evaluating a rule overrides the results of the rest. Furthermore, XACML provides a rich set of functions to compare strings, numbers, dates, urls, and so on which allowed us to specify temporal conditions such as in P2 (whereas EPAL failed to do so).

Similarly to EPAL, it is difficult to specify exclusivity and exceptions with XACML (i.e. "only cardiologists" (P1) and "hospital staff other than physicians" (P6)). Assumptions must be also made on whether the access to a resource is prohibited unless it is explicitly granted; XACML (as far as we could tell) is silent on these assumptions. However, XACML supports denying groups and individuals access to resources. If explicit denial is required, scalability issues will arise when a domain has hundreds of roles. For example, if only physicians are allowed to modify health records, then access prohibition policies ought to be specified for nurses, interns, administrative staff, and every healthcare personnel other than physicians; the result is extremely long and complex policy specifications. In addition, XACML supports negative rules—however not appropriate for describing P1—where a group member is denied access to a resource, permission which he/she would be otherwise be granted given the access rights of the group. These kind of exceptions, although supported, are not recommended given that they could lead to policy violations.

XACML is closely coupled with implementation details, making assumptions on how the data is stored. XACML supports setting xPath expressions as attribute values, therefore assuming information is stored as XML documents (e.g., XML-stored health records). Although supporting xPath adds to the richness of the language, it sacrifices the flexibility to use other types of data storage and system implementations. This is one of the reasons we

did not select XACML in our approach. XACML is not only a policy specification language, but a more integral component and an active participant in servicing data access requests in particular implementations.

Another issue with XACML is its verbosity; there is more boilerplate code and long XML namespace declarations than domain specific concepts that actually describe policies. Without tool and user interface support for specifying and reading XACML policies (to perhaps non-technical administrative employees), the language by itself is unusable for describing complex policies. The same applies to EPAL.

Both EPAL and XACML have not been commercially adopted. Some argue that XACML was designed for monolithic organizations, failing to meet the needs of federated enterprises with independent system users and distributed deployments [82].

Cassandra

Cassandra is both a high-level, declarative policy language with formally defined semantics and a supporting trust management system [37]. Cassandra supports credential- and role-based authorization to resources. In addition, Cassandra can refer to remote policies, therefore it is appropriate for large-scale distributed systems.

Cassandra is based on Datalog with constrains, therefore it depends on a constraint domain C—a first order language supporting, at least, boolean values and the identity predicate (=). Cassandra's rules include parameterized roles, parameterized actions, and predicates. A rule is formed from a prefixed head predicate, one or more prefixed body predicates, and a constraint expression that is an element in C: $E_{loc} \diamond E_{iss}.p(\vec{e}) \leftarrow loc_1 \diamond iss_1.p_1(\vec{e_1}), ..., loc_n \diamond iss_n.p_n(\vec{e_n}), c$, where p_i is a predicate name while e_i is an optional expression tuple. Cassandra predicates may specify a rule issuing party (E_{iss}) and a location (E_{loc}). A policy guiding access control decisions is constituted by one or more rules.

Cassandra can also express a credential $E_{loc} \diamond E_{iss}.p(\vec{e}) \leftarrow c$, where entity E_{iss} issues a certificate asserting $p(\vec{e})$ which is owned and stored by E_{loc}. This property is very important for decentralization since "access control in large-scale heterogeneous distributed computing environments is fundamentally different from access control in a single administrative domain"; in the former, system users may be unknown.

Cassandra's syntax includes pre-defined predicates to define role membership, activation, and permissions: `permits(e,a)` (entity e may perform action a), `canActivate(e, r)` (entity e can activate role r), `hasActivated(e, r)` (entity e has activated role r; indicates who is active in which role), `canDeactivate(e1, e2, r)` (entity $e1$ can deactivate $e2$'s role r), `isDeactivated(e, r)` (deactivates e's role r), and `canReqCred(e1, e2.p(`$\vec{e}$`))` ($e1$ is allowed to request/receive credentials asserting $p(\vec{e})$ issued by $e2$; specifies the conditions for disclosing a credential). For example, the rule `Alice`$\diamond$`UCam` $\leftarrow$ `canActivate(Alice, Student(Math))` asserts that Alice is a Math student. Through the `canDeactivate` predicate, Cassandra supports cascading permission revocation.

Cassandra has a strong focus on the activation and deactivation of roles. Access to a resource—granted through the `permits(e,a)` predicate—is conditional on the activation of an associated role; a permission denotes a specific trust relationship (e.g., consenting to access an electronic health record through the `Treating-physician()` role). Also, Cassandra's advantage is its consideration for decentralization given the pre-defined predicates for trust management and credentials exchange between parties (i.e. `canReqCred` predicate). For instance, the case study addressed by the authors is a national Electronic Health Record system, where policies are complex given the nature of the domain. In contrast to EPAL and XAMCL, it is not particularly concerned with user attributes nor on the properties of the accessed resources.

Although the core language is small, it can gain expressiveness through expanding the constraint domain C, therefore through user-defined, domain-specific functions and predicates.

This flexibility allows to custom tailor policies for specific domains and organizations. For example, the domain-specific rule

```
isStudent(name, college, subject)} ← matriculated(name,matricDate)
```

states that a person is a student if he/she is matriculated. However, this flexibility comes at the price of potential disagreement in the understanding and interpretation of policies, the burden of specifying the policy syntax and semantics, and potentially an obstacle to interoperability.

In Cassandra, the meaning of "role" is more embracing than the traditional meaning of a role—a position occupied within an organization and the responsibilities and rights appointed to it [103]. A role in Cassandra is a subject attribute; activating such role is equivalent to setting the attribute. Examples are `Register-patient()`, `Express-consent()`, and `Access-denied-by-patient()`—predicates which otherwise may be interpreted as actions such as `Read-record-item(pat,id)`. While the authors choice of a more broad role definition is deliberate given that role activation has an observable effect on the state of their operational model, the semantics of a role remain counterintuitive and confusing.

Alike XACML, the language in Cassandra is closely tied to implementation details—recall that Cassandra is both a language and a system. The language's syntax is focused not only on expressing natural language policies but on how the policies are implemented and enforced by the system, even if the policy is physically separate from the application code. This tight coupling has both benefits and drawbacks: the benefit is straightforward integration with the implementation through shared syntax and semantics; the drawback is that it is less clear as a standalone language since it is biased towards how the authorization model is operationalized, thus less useful with other implementations.

In our experience, it was difficult to naturally express the policies in table 5.1 with Cassandra; often policies had to be rephrased or molded to Cassandra's semantics. For example the

policy "no one shall have the ability to delete clinical information until the appropriate time period has expired (6 years)" was rephrased to "clinical information can be deleted after the appropriate time period has expired (6 years)". However, the rephrased policy could be incorrectly interpreted as "anyone has the right to delete clinical information after 6 years". As previously mentioned, it is difficult in Cassandra to refer to resource properties with the provided pre-defined predicates, unless it is in relationship to a subject.

Despite these disadvantages, a wise design decision (and where EPAL and XACML stay silent) is not supporting prohibition or negation; unless explicit rights are given to access some resource or execute an operation, everything is prohibited for everyone. Not supporting negation results in more simple, unambiguous, and compact policies, and resolves the complexity of conflict resolution of contradictory policies.

PeerTrust

PeerTrust is a policy language to support run-time trust negotiation and access control to sensitive resources in peer-to-peer architectures [118][200]. PeerTrust enables establishing trust between parties by supporting the exchange of information to proof the identity and trustworthiness of parties to each other. Exchanged data are digital credentials that may be endorsed or "signed by" third parties. Trust negotiation, for instance, involves an iterative process of credential exchange until both parties are satisfied and have enough reasons to trust each other.

The language is based on first order Horn rules $lit_0 \leftarrow lit_1, ..., lit_n$, where lit_i is a predicate with 0 or more arguments. For example, a person may access an electronic health record if he/she is a doctor: `access(ehr, X) ← doctor(X)`. In addition to this basic rule for composing policies, the syntax lit_i@$Authority$ supports specifying certifying authorities, where authority $Authority$ asserts the predicate lit_i. For example, a hospital can certify that

X is a doctor: `access(ehr, X) ← doctor(X) @ "hospital"`. Therefore, the "hospital" authority is in charge of evaluating the predicate `doctor(X)`. The authority argument can be nested such that lit_i@$Authority1$@$Authority2$ means that Authority2 can assert that Authority1 asserts lit_i. For example, a university may implicitly enable a student to certify his/her student status to a third party by supplying a university-issued student id. The assertion or denial of such predicates is a key feature for establishing trust-based relations.

PeerTrust assumes an environment where peer A sends a query to peer B; peer B'S response depends on who A is. $\$context_j$ within the syntax lit_i@$Authority\$context_j$ enables, for example, specifying the author or requester of a given query. For example, in the policy `access(ehr, X) $ Requester = X ← Requester = "Dr. Brown", $ Requester = X` refers to the author of query `access(ehr, X)`. This feature of the language can be leveraged to limit the authority to a particular query. In this example, the predicate `Requester = "Dr. Brown"` states the condition that the requester to access the required EHR must be Dr. Brown.

As mentioned above, PeerTrust rules enable the interaction and the establishment of trust between autonomous peers for the controlled access to resources or the execution of actions. In our example, peer A sends the request `access(ehr, Dr. Brown)` to peer B; B enquiries (to itself given `@"hospital"`) whether Dr. Brown is a doctor: `doctor(Dr. Brown)`. If the predicate is true, then access to the record is provided.

Through a logic-based approach, PeerTrust allows expressing a wide variety of policies based on predicates. In this aspect, its syntax and semantics are very similar to those of Cassandra, both relying mostly in application-specific predicates. PeerTrust is, however, particularly focused on credential exchange and third party certification to initiate a trust relationship; although access to resources can be described, it is not emphasized. Consequently, PeerTrust literature presents few examples and guidelines with respect to service access control.

In addition, PeerTrust does not explicitly show how to describe environment and temporal conditions such as the current date; our solution for expressing P2 (showed in table 5.1) was an ad-hoc one, however without the certainty of having expressed the corresponding predicates correctly (pertaining to EHRs expiration date).

PeerTrust does not support (or is silent on the topic of) negation, prohibition, exceptions, or explicit exclusion, making it difficult to describe many policies. Also it does not explicitly support logical conjunction nor disjunction, therefore, in P8, we had to generalize `guardian(X,Y)` where the policy reads "custodial parent or legal guardian".

Ponder

Ponder is a declarative, object-oriented policy language for distributed objects systems; policies are associated to a single or a group of objects [84]. Policy typing, instantiation, and inheritance is supported. in Ponder, a policy "is a rule that defines a choice in the behavior of a system", is independent from the system's implementation, and supports the dynamic modification of the system's behavior.

At a minimum, a Ponder policy refers to: (a) a "subject"—the user or principal (human or software); (b) a "target"—the object that the subject wants to access, and; (c) an "action" to be performed on the target. Subjects or targets of a specific type can be grouped in a "domain" such as `/employees`. Additionally, time- or attribute-based constraints can be imposed on policies through the `when` keyword. Constraints can also be enforced on a group of policies (or meta policies) which are specified with a subset of the Object Constraint Language (OCL).

For example, a simple authorization policy such as "a doctor who is a hospital's employee is authorized to treat hospital patients" is specified:

```
inst auth+ Doctor{
  subject /hospitalEmployee/Doctor;
  target  /hospitalPatients;
  action  treat());
}
```

Ponder supports access control to resources or services through authorization (positive and negative), delegation, information filtering (transforms an action's input or output according to some condition), obligation (actions which must take place triggered by an event), and refrain policies (actions that subject's must refrain from making). These policy types have a well-defined syntax that guides the policy maker in clearly and flexibly defining organizational policies.

Lastly, Ponder allows to specify composite policies through grouping, role association, relationships, or policies based on management structures. Groups of policies can be associated by being applied to the same target or relate to the same department or action. Roles policies associate policies that are performed by the same type of subject (e.g., manager, physician). Role policies inheritance is also supported—the role manager inherits as well the policies of role employee. Relationship policies associate different subject roles through rights and obligations towards each other, or towards common target resources.

In summary, Ponder is a language whose main goal is specifying access control polices stating the activities that a principal can perform with respect to a target. It is good language choice when the goal is protecting services and information from unauthorized access. Ponder supports both positive and negative authorization policies as well as exclusions, therefore making it straight forward to specify policies such as P1 where only one type of member within a domain is authorized to perform an action.

Ponder is a very rich and well-defined language, and it is very clear what is defined by the

language and what is domain specific (defined by the policy maker). This trait makes it easier to specify policies and prevent ambiguities. In addition, actions are associated directly with object methods (with reference to the OO programming style), so from an implementation standpoint it makes it easier to translate policies to system behavior. This, of course, has the drawback of being bounded to a particular programing language or paradigm.

In our experience specifying the evaluation policies, run into the difficulty of not begin able to express the relationship between more than two entities (subject and target) in Ponder. For example, in P3, the subject is the doctor and the target is a health record, but there where no means to include the patient that the record belongs to; it may be desirable to grant access to this patient's health record according to the attributes of the specific patient. Instead we are constrained to refer more generally to a "patientEHR". Similarly, we failed to specify P5, given the complicated association between the illness domain that the drug is meant to cure and the physician's specialization. Ownership relationships and other more complex associations between entities are difficult to specify in Ponder.

Rei

Rei is a language based on deontic logic—policies are expressed in terms of rights, prohibitions, obligations and dispensations (obligation waivers) [147]. The version of Rei we evaluate is implemented in Prolog; alternative versions of Rei are described in semantic web languages. According to the authors themselves, the logic-based notation is far more expressive than the DAML+OIL and OWL alternative implementations [148]. Alike previously discussed policy languages, Rei itself is domain independent, but depends on a vocabulary of domain classes and their properties (e.g., patient, body temperature) in order to specify meaningful policies.

Rei supports three types of objects: (a) policy objects described as `PolicyObject(Action`

,Conditions), where a PolicyObject may be a right, obligation, prohibition, or a dispensation; (b) meta-policies which enable conflict resolution by describing policies precedence or priority; (c) speech acts that describe policy delegation, revocation, and canceling, and requests for an action or a right.

A policy is a set of rules. Rules are associated to subjects in the form of has(Subject, PolicyObject), where a subject is an specific entity or a class of entities (e.g., Dr. Brown or physicians). Recall that a policy object stands for a deontic concept, an action, and a condition. Therefore, expressing the policy "an individual has the right to print if he/she is an employee" can be expressed as:

```
has(X, right(printAction, (employee(X))))
```

If the subject is Joe, the same rule is expressed:

```
has(Joe, right(printAction, (employee(Joe))))
```

Rei provides further expressiveness to describe an action through specific parameters: action (ActionName, TargetObjects, Pre-Conditions, Effects). For example, the printing action with the pre-condition that the printer having both a cartridge and paper can be represented as:

```
action(printOnePageHP, [printerHP], (containsCartridge(printerHP),
    availablePaper(printerHP, X), X > 1),availablePaper(printerHP,
    X-1))
```

Rei also provides syntax and semantics for action cardinality and order: seq(A,B) (action A happens before B); nond(A,B) (either action A or action B can take place); repetition (A) (action A can happen multiple times); once(A) (action A can happen only once). Rei also allows constructing more complex conditions through the logical conjunctions and and or, and the negation not.

Rei is a compact, well-defined and expressive language for describing business policies. Alike some analyzed languages, Rei is silent on whether all actions are prohibited unless they are explicitly authorized. However, the presence of the `prohibition` PolicyObject suggests that prohibitions must be explicitly stated as well as rights. For example, it is not straightforward to express the constraint that "only cardiologists are allowed..." in P1. As a workaround, the "cardiologist" class of users can be authorized to access cardiac EHR data and simply assume others are forbidden of such access. A third alternative is to express a prohibition policy from which users of type "cardiologists" are excluded.

Also, it is difficult to express policies such as "subject A requests access to X which belongs to or is part of subject B". `TargetObjects` within `action(ActionName,TargetObjects,Pre-Conditions,Effects)` allows specifying an entity as the target of the action. However, the policy maker may wish to specify a predicate that refers to a specific type of object. For example, in policies P3 and P8 we may instead want to refer to `ehr(B)` as a `TargetObjects` to refer to a specific patient's health record. Similarly, we may want to "pre-process" an object before granting authorization to it. For example, in P5 we can express `anonymized(EHR)` in order to refer to the set of EHRs deprived of personal identifiers.

Lastly, another difficulty was the lack of arguments to Rei's actions. This can be clearly observed with policy P4; instead of action `prescribeDrug` we may want to express an action along the lines of `prescribe(Drug)`, where Drug is a variable that can be referred to in the body of the action's conditions (Pre-Conditions).

5.2.3 Evaluation summary

Individual strengths make these languages good candidates for expressing healthcare policies. For example, EPAL and XACML are very expressive and can describe policies in great detail (also due to the extensibility of XML). Cassandra allows to parametrize roles and

actions, and therefore can specify more nuanced policies. In addition, rich policies can be flexibly expressed in Cassandra through user-defined constraint domains, and thus by way of domain specific predicates. Finally, Cassandra provides the ability to specify the location and issuer of policy predicates, therefore supporting decentralization and collaborative policy description. PeerTrust instead responds to user queries depending on the identity of the requester and the trust built through a sequence of credential exchanges. Ponder allows grouping subjects and targets hierarchically by domain, increasing domain knowledge and gaining understanding of organizations asset and personal structures. Ponder also offers tool support for describing policies. Lastly, Rei's compact syntax based on the solid logic fundaments of Prolog allows readability even without tool support. Common to all these approaches is the definition of a subject, an action, the target object, and a set of conditions within a policy.

Despite these strengths, no language can fully specify the set of evaluation healthcare policies (as observed in table 5.2)—mainly EHR access control policies—due to divergent purposes and design decisions embedded in these languages. EPAL and XACML focus on fined-grained attribute-based authorization to data. Cassandra and PeerTrust emphasize on trust management and the exchange of credentials. Ponder describes policy hierarchies through policy typing, inheritance, and instantiation as an access control approach. Rei has deontic foundations, thus describes policies in terms of rights, prohibitions, obligations and dispensations. These different goals evoke significant differences in syntax and semantics. A common disadvantage of most of these policy languages is the ability to describe an action's target as "x which belongs to y"—for example, "Dr. Brown (subject) can read (action) Joe's electronic health record (target)".

Given these differences in language semantics, goals, advantages, and disadvantages, the choice of policy language is application- and organization-specific, thus according to which is best suited for the task at hand. As a result of this analysis, we have chosen to use Rei to

	P1	P2	P3	P4	P5	P6	P7	P8
EPAL	✓	✗	✓	✓	✓	✓	✓	✓
XACML	✓	✓	✓	✓	✓	✓	✓	✓
Cassandra	✓	✓	✓	✓	✓	✓	✓	✓
PeerTrust	✓	✓	✓	✓	✓	✓	✓	✓
Ponder	✓	✓	✓	✗	✓	✓	✓	✓
Rei	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.2: Evaluation privacy policies languages

describe policies in healthcare, our sample domain. We have chosen Rei for its simple, clear, compact, and flexible logic-based syntax and semantics. Although Rei failed to successfully describe some policies, we will assume from now on that everything is forbidden unless it is explicitly stated. This solves the difficulty of expressing policies such as P1—"Only cardiologists..." is rephrased as "Cardiologists are allowed...". In the process of formalizing natural language policies, we realized that reasonable rephrasing (without changing the semantics of the policy) is necessary to make more precise privacy statements and produce correct formal policies.

Although Cassandra is also a logic-based language with properties such as compactness, we chose not to use it for two main reasons. First, it is heavily focused on the activation and deactivation of roles which can be easily expressed in Rei. Second, in Cassandra there is no clear distinction between actions and roles, making it hard to write policies and understand.

We decided not using XML-based languages—XACML and EPAL—due to their verbosity and complexity; without effective tool support XML-based policy descriptions are hard to manage. However, the widespread adoption of XML, available processing tools and language bindings, increased interoperability, and expressiveness are unarguable benefits of choosing semantic web notations [253].

The purpose of PeerTrust diverges from our objectives; we are not particularly interested

in the iterative exchange of credentials. PeerTrust, however, may be appropriate as a Web-of-Trust-like mechanism for establishing trust between parties. Although we are concerned with trust relationships, this initial stage in which parties gain this trust is out of the scope of this research.

Although Ponder is a concrete and well described language, we exclude it from our work since it does not provide the flexibility to define other types of authorization properties and relationships besides the ones predefined in the language syntax. This is in part due to being an object oriented language where classes have predefined properties, and thus it is difficult to add more properties without creating new classes. Ponder, therefore, was unable to express P4 which exhibits more complicated relationships between objects and attributes. Also, it appears that rules only allow a single constraint-expression (condition), hence authorizations that are contingent on multiple conditions cannot be expressed in a simple way.

The goal of selecting a policy language—Rei as chosen by way of this evaluation—is ultimately binding internal privacy and operational policies to an organization's offered services in order to guarantee compliance. In section 6.1.1 we describe how we leverage Rei's concepts to describe domain-specific privacy and operational policies, how policies are thereafter evaluated in order to make access permission decisions (section 6.1.2), and how they are implemented within a software system (section 7.3).

Chapter 6

Policy-Based COAST Services

Several key insights motivate our approach. First, individuals and organizations are autonomous, independent, act in their own self-interest pursuing their individual goals, and interact upon specific trust relationships. Therefore, accommodating decentralization is essential. Parties must be able to effectively offer, revoke, use, and compose services that belong to independently managed organizations.

Second, given that these parties have different relationships and trust is not homogenous, it is an important requirement to protect an organization's assets to differentiate among service consumers. This requires sophisticated fine-grained authorization and security mechanisms so that users are provided with bounded custom capabilities according to privacy policies defined by the service provider or by the individual whom the data describes, or by both.

Third, services ought to be tightly coupled with organizational policies to prevent privacy violations. It is rather common that privacy and operation policies are implicitly embedded in the culture of an organization or documented within static documents. These policies are usually divorced from systems'—and the provided services—behavior, opening up privacy breach and insiders' abuse opportunities. Therefore, services that are user-specific need to

be created according to and bound to policies.

Fourth, security is an imperative given that data and computation cross agency boundaries. Differential access itself calls for security provisions—parties may maliciously or accidentally attempt to access information and functionality which has not been conferred to them, attempt to execute malicious code, or give the granted capabilities inappropriate uses.

Lastly, given the myriad uses and users of a service, it is unlikely for service providers to tailor services to meet the diverse current and future needs of all users. Therefore, the burden of service customization needs to lie on the user and not on the provider.

These insights lead to a novel approach towards web-based services which have two fundamental properties: first, services are personalized by the provider for particular users or user types according to explicit and well understood privacy policies to specialize offerings for their array of users, and at the same time protect the provider's assets (information and algorithms). Second, services are customizable—they allow users to leverage the available capabilities to compose new services. These properties tackle the weaknesses of current web services: rigidness, uniformity, and generalization. In short, the principal goal of our work is to enable service providers to dynamically create, based on a set of formally defined privacy policies, user-specific services which can be customized by service consumer applications.

Figure 6.1 captures the main idea of our approach: a service provider maintains, on one hand valuable and protected domain information, and on the other hand the relevant privacy and operational policies that regulate the access to this data. Based on these policies, user-specific services (e.g., service client 1, service client 2, and service client 3) are dynamically created, each with a bounded set of capabilities (i.e. functions, e.g., $fa()$) and potentially with access to some domain data. These bestowed privileges are coherent with a principal's individual authority as prescribed by the policies. Service consumers (e.g., client 1, client 2, and client 3) use their corresponding services by way of capability URLs (introduced in 5.1)

by sending messages to their services, messages which may contain custom closures (e.g., $\lambda(a(b(x)))$) to be executed by the service for a desired result information (a response is sent back to the user) or side effect (the execution of the service has some effect in the state of the overall system such as the modification of the domain information).

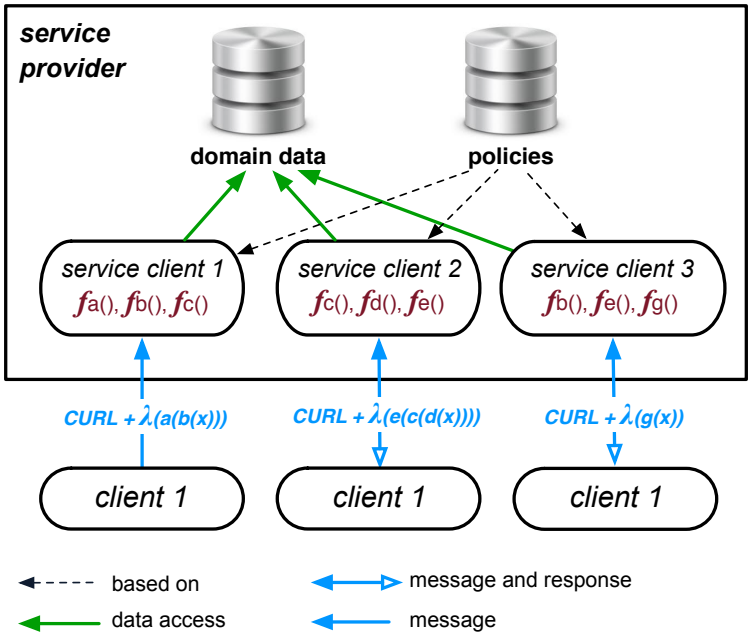

Figure 6.1: A provider, based on a set of policies, dynamically creates user-specific services.

In this context, an information service provider is an individual and decentralized system managed by a single island (in COAST parlance) and which has a set of capabilities and information to offer to distinct service users. This individual system/island may exist within a larger organization and interact with other organization-wide islands. In section 7.6 we elaborate in the role of a provider within a system of systems in the context of COASTmed.

In the following sections, we describe in more detail the constituent elements of this approach. In concrete, we describe how policies are specified and enforced for individual users, how service capabilities are specified and bound to formal policies, how services are dynamically created for specific users, and ultimately how users leverage service capabilities to compose custom services.

6.1 Policy-based Differential Service Provision

One of the main goals of this work is permitting service providers to deliberately restrict the access to their information and algorithms. This capability enables providers to differentiate among their users for purposes of both access control and service specialization. Leading to policy-based service provision are the formal specification and evaluation of policies, the specification of user services, the associations between policies and service capabilities, and lastly the dynamic creation of user-specific services and their corresponding CURLs.

6.1.1 Specifying Policies

Privacy-aware services provide patient data solely to authorized parties with distinct access privileges. To guide the behavior of these services, it is necessary to capture organizational policies describing the conditions under which data is disclosed. Policies thus dictate how systems and users ought to behave so that processes and individual activities involving individuals, organizations, and data are in compliance with the defined policies. We focus on privacy and operational policies related to the disclosure of personal information in support of policy-based access control. Although operationally they are treated equally, there is a semantic distinction between privacy and operational policies; the former are put in place with the goal of protecting agency assets from unauthorized intruders while the later support the division of responsibilities within an organization. For example, a privacy policy may prevent interns from accessing customers' information, while an operational policy may dictate that only senior nurses and doctors are responsible for discharging a patient.

Using Rei to specify policies

Formal policies not only disambiguate natural language, but also allow automated processing. We have selected the Rei policy language to specify policies, after evaluating a set of languages which included EPAL, XACML, Cassandra, PeerTrust, and Ponder (section 5.2). The criteria of evaluation included the ability to express subjects, roles, data objects, attributes, obligations, rights, prohibitions, conditions, exceptions, use purpose, and temporal constraints. Rei was selected for its expressiveness, simplicity, flexibility, and compactness.

To simplify policy specification we restrict and modify Rei's syntax and semantics to better suit our goals, namely policy-based access control to services. For instance, since we are only concerned with a principal's authority to access a resource—i.e. with a principal's *right*—we exclude both the *obligation* and the *dispensation* policy objects. In addition, we adopt Cassandra's assumption that everything that is not explicitly allowed is forbidden—a "closed world assumption"—therefore disallowing the *prohibition* policy object. The unavailability of prohibitive policies significantly simplifies a policy document, lessening the probability of conflicting policies. We also differentiate between direct and indirect target objects, since we want to be able to express policies such as "*the nurse has the right to dispense IV therapy to patients*", where *IV therapy* is the direct target object and *patients* is the indirect target object. Lastly, we exclude from our constrained version of Rei the concepts of action preconditions and effects which are, at this stage, unnecessary for this access control approach. Therefore, the contents of a policy in our simplified version or Rei is as shown in figure 6.2, where "the basic components of any authorization policy are the actor, action and target" [144], as well as conditionals on those authorizations.

policy(subject, action, direct target, indirect target, (conditions))

Figure 6.2: A policy's contents.

Policies bestow rights according to zero or more conditions. These are domain- and organization-

specific, and are defined by the service provider. For example, a policy stating that only cardiology specialists are allowed to modify the portion of medical records pertinent to cardiac ailments may be specified in our version of Rei as:

```
policy(physician,update,cardiac_ehr,patient,(instanceOf(*physician
    *,cardiologist)))
```

In the previous policy, the condition `instanceOf(*physician*,cardiologist)` requires the subject in the `*physician*` role to be a cardiologist. Thus policy conditions establish the relationships between objects, entities, or concept. In the previous example `instanceOf` describes a relationship between *physician* and *cardiologist*. Policies can refer abstractly to subjects and objects by referring to roles and object categories. For example, in the previous policy, `*physician*` refers to a principal's role.

To implement policies, we use a delimiter symbol "*" to indicate that `*physician*` is a variable. In other words, abstract conditions include variables, for whose evaluation it is necessary to provide real values to these variables. In addition, Rei's conditions are operationalized as predicates—a natural approach given Rei's functional nature—namely functions that return boolean values. For example, the condition predicate `availablePaper (printerHP, X)` stating that "the available paper in the printerHP printer is X" returns *true* or *false* when a value is assigned to `X`.

Types of policies

We identify three types of policy conditions:

(a) dependent to a principal's role, identity or association;

(b) dependent on environmental conditions such as time and date, and;

(c) contingent upon the target (i.e. argument) of an action.

An example of a condition based on a **principal's role** is when the access to an electronic health record is conditional on a person being in the attending physician role: `instanceOf(*person*, attending physician)`. Our approach, however, not only subsume role-based access control, but policies can also refer to specific principals in support of identity-based access control (IBAC). A condition upon the **principal's identity** instead provides fixed values to conditions related to principals as opposed to variables. For example, `instanceOf (Andrew Jones, attending physician)` is no longer an abstract condition, but acts upon the identity of a principal. However, based on experience, associating permissions to roles is preferred, since it is known that individuals-based policy specification does not scale [203][37]. Also, conditions can be related to a principal's association to a group. For example `department(*person*, cardiology)` states that a person is required to be a member of the cardiology department.

Context or environment conditions are those which restrict access to service data and computation based on endogenous factors such as the time of day or the date. For example, a policy that states that a medical intern can only access the EHR database between 8:00 A.M. and 6:00 P.M. may be specified as:

```
policy(intern,access,EHR,patient,(after(*currentTime*,7:59),
   before(*currentTime*,18:01)))
```

There are other interesting environmental conditions that can be exploited, such the outside temperature. For example, through a COAST-enabled controller, a heating system can be turned on according to the policy

```
policy(person,turnHeatOn,null,null,(lower(*currentTemp*,65))))
```

A policy which depends on the **direct and indirect targets of an action**, for example, may allow a physician to update a patient's electronic health record if and only if the physician is the patient's primary care physician:

```
policy(physician,update,EHR,patient,(primaryCarePhysician(*
    physician*,*patient*)))
```

A policy may include and combine all these described type of conditions; there is no constraint on the number of conditions a policy may have.

Negation and alternative conditions

Conditions may also be negative, which is useful when specifying exclusions in a policy. For example, a policy may state that residents may perform supervised procedures unless they are junior residents. This is accomplished in Rei through the negation *not*:

```
policy(resident,perform supervised procedures,null,null,(not(
    instanceOf(*resident*,junior resident))))
```

Policies may also include alternative conditions, namely a privilege is bestowed if at least one within a set of alternative conditions is true. Rei's logical conjunction *or* allows specifying alternative conditions. For example, a policy may state that a physician may prescribe aspirin if the patient is in pain or if he requires a blood thiner.

```
policy(physician,prescribe,aspirin,patient,(or(diagnosis(*patient
    *,pain)), (diagnosis(*patient*,poor blood flow))))
```

Through direct and indirect policy objects we can provide more complex semantics on the relationship between subjects, actions, and objects, and to refer to an object in reference to another (i.e. to whom and for whom relationships). So in the prior policy, the subject is the `physician`, `aspirin` is the direct target, and the `patient` is the indirect target.

Policies and ontologies

Although Rei is domain independent, policy specification relies on domain vocabularies to capture domain rules and knowledge. These vocabularies increase shared understanding of policies within organizations and domains. For example, the policy:

```
policy(physician,prescribe,fluoroquinolones,patient,(diagnosis(*
    patient*,anthrax)))
```

states that a physician can prescribe fluoroquinolones if the patient has been diagnosed with anthrax. Both `fluoroquinolones` and `anthrax` are terms within the ATC code and the ICD-10 healthcare ontologies correspondingly.

Resolving conflicting policies

One of the challenges surrounding policy specification is addressing policy conflict. Much of the work addressing this issue provide algorithms to resolve conflicting permission and prohibition policies. For example, Uszok et al. provide an algorithm to detect and harmonize positive vs. negative authorization policies, positive vs. negative obligations, and positive obligation vs. negative authorization [257]. Sujansky et al. address similar challenges where one rule permits access to a data type while another rule denies access to an instance of the same data type [246]. These algorithms consist in, for example, evaluating policies in order, ignoring subsequent, potentially conflicting related policies as done by the EPAL language [17]. In addition, EPAL defines a default ruling (decision) for when no rule is related to a specific request. XACML has available different combining algorithms, such as prioritizing "deny" or "permit" decisions, choosing the "first applicable" rule (such as done in EPAL), and granting access if no rule returns a "deny" decision [203]. Rei, for instance, leverage meta-policies to resolve modality conflicts (i.e. positive vs. negative or right vs. prohibition) or to establish policy priority. Similar precedence or priority approaches are addressed by

Jaeger et al., who deal with conflicts in "access control spaces"; the purpose of these algorithms is to replace "ad hoc policy modifications to resolve conflicts" [142]. Kamoda et al. present the *free variable tableaux* conflict detection method, which translates policies to logical sentences to detect conflicts between authorization and obligation policies, propagation and action composition policies, Chinese wall policies, separation of duty policies, and time constraint policies [150]. Samarati et al. present an overview of policy conflict resolutions methods (e.g. denials-take-precedence, most-specific-takes-precedence, most-specific-along-a-path-takes-precedence, strong/weak, priority level, positional, grantor-dependent, time-dependent) [229]. Even a conflict database has been proposed to characterize the types of conflicts that may arise while combining policies [94].

In our approach, we do not have to deal with this type of conflict resolution, since, by design principle, we do not allow prohibitive policies—we assume everything is forbidden unless explicitly authorized. This simplifies policy specification and increases security by minimizing the opportunities for unauthorized access. Also to our advantage is that our modified version of Rei is based on propositional logic, and therefore is decidable and complete [113]. In other words, we can determine the truth value of a propositional formula using a truth table, therefore no policy evaluation result (having all the required information) is undecidable.

However, we can solve for redundant policies based on roles (although these are not considered a case of conflicting policies). Consider for example the policies

```
policy(employee,scheduleVacation,null,null,null)
policy(physician,scheduleVacation,null,null,null)
```

which bestow the right to schedule a vacation to both employees and physicians. Since a physician is a type of employee, this policy is redundant. To solve this, we can discard the policy appertaining to the sub-role, since the sub-role is included in the super-role. It may be however desired to keep both policies, since in the future, a policy maker may decide to

withdraw this right to employees with the exception of physicians. This decision is domain and organization specific.

A conflict situation is, however, when trying to add a policy whose subject, action, and target already exist within the policy document. One option is not to allow this addition but suggest the user to modify the existing policy. A second option is to let the policy maker choose between these two policies. A third option is to merge the two policies by simply appending the list of conditions of the new policy to the list of conditions of the existing one such that

```
policy1(physician,readEHR,patient,null,(inDepartment(*physician*,
    traumatology), name(*physician*, John)))
```

and

```
policy2(physician,readEHR,patient,null,(primaryCarePhysician(*
    patient*, *physician*)))
```

become

```
policy3(physician,readEHR,patient,null,(inDepartment(*physician*,
    traumatology), name(*physician*, John), primaryCarePhysician(*
    patient*, *physician*)))
```

However, we need to address conflicting policy conditions for a given subject-action-target tuple. Consider this example provided by Kim et al. [158]

P1: (BusinessPartner, ((Read, OrderInfo), Research, CurrentTime=5PM-11PM, 0))

P2: (BusinessPartner, ((Read, OrderInfo), Research, CurrentTime=11PM-9AM, 0))

where policies conflict on condition *CurrentTime*. There are three possible solutions to resolve this conflict: first, reject the addition of this new policy altogether and send a mean-

ingful error message to the policy maker; second, allow the policy maker to choose which to keep among the conflicting conditions; third, merge the conditions such that the conflicting predicates in the previous example resolve into the predicate *CurrentTime=5PM-9AM*. The problem with the third option is that not all conditions will be resolvable in such way. Therefore, the best option in this case is to allow the policy maker to make a deliberate choice between the conflicting conditions.

There are instances, however, where semantically equal predicates are not necessarily in conflict. Consider for example the policy

```
policy(employee,access,privateDocument,null,(inDepartment(*
   employee*,accounting), inDepartment(*employee*,auditing)))
```

where the two `inDepartment` conditions are not in conflict, but together specify that an employee must be part of both the accounting and the auditing department to access a document `privateDocument`. Since it is difficult for a software program to guess the intent of the policy maker, the most sensible solution is to allow the policy maker to choose to include one or the other, or both conditions in the resolved policy.

6.1.2 Policy Evaluation

In order to assess if a principal is entitled to access a set of computation capabilities and information, it is necessary to evaluate the policies that are relevant to the principal. The evaluation process relies on knowledge representation, where policies and their conditions include relations described as facts and rules. As previously described, an abstract policy is composed of variable and constant values. Evaluating a policy involves replacing variables with constants and verifying that the policy conditions are met. Conditions are operationalized as predicates (functions which return a boolean value), whose execution, given a set of arguments, return *true* or *false*. For example, we can evaluate the policy:

```
policy(physician,update,EHR,patient,(primaryCarePhysician(*
    physician*,*patient*)))
```

by providing real values to the variables *physician* and *patient*:

```
policy(Dr Jones,update,EHR,John Smith,(primaryCarePhysician(Dr.
    Jones,John Smith)))
```

This policy's predicate evaluates to *true* if and only if Dr. Jones is John Smith's primary care physician, therefore granting Dr. Jones access to the update EHR capability with respect to that particular patient. To assert the relationship between Dr. Jones and John Smith, the availability of a fact base is necessary.

For permissions to be granted to a principal, all the policy conditions need to evaluate to *true* and at least one within a set of alternative conditions, if any, needs to evaluate to *true*. In other words, the result of a policy's evaluation is the result of the logical conjunction of all its conditions' truth values. For example, the policy

```
policy(person,submitTimeSheet,null,null,(equal(*currentMonthDay
    *,1),or(role(*person*,intern),role(*person*,contractor))))
```

can be translated to propositional logic. Let `right(submitTimeSheet)`, `equal(*currentMonthDay*,1)`, `role(*person*,intern)`, and `role(*person*,contractor)` be the propositional variables R, C1, C2, and C3 correspondingly. Then, this policy can be translated to

$C1 \wedge (C2 \vee C3) \rightarrow R$

This means that both C1 and $(C2 \vee C3)$ need to be true for R to be true. For $(C2 \vee C3)$ to be true, at least one of C2 and C3 needs to be true. Therefore, a person can submit a work time sheet if the day of the month is the 1st (C1 is true) and if the person is an intern or a contractor (either C2 or C3 is true or both).

Given that the first step to evaluate a policy requires replacing variables with constants, it is necessary to provide the evaluation function with the policy and a list of key-value pairs of variable names and their values (e.g., var-name-1, value-1, var-name-2, value-2 ...). If all of the values to replace their corresponding variables are provided, then the policy evaluates either to *true* or *false*. On the contrary, if one or more variable values are not provided, the policy is not evaluated and instead the *'not-evaluated* value is returned.

In section 7.4.1 we provide an in-depth description on how policies and conditions are operationalized, how facts are obtained, and how policies are evaluated in practice in the context of COASTmed.

6.1.3 Associations of Policy and Service Capabilities

User-specific services are differentially provided to diverse stakeholders according to an organization's policies. For doing so, a tight coupling between policies and the organization's capabilities—namely, its COAST islands' bindings)—is necessary. For example, a policy

```
policy(person,book-OR,null,null,(instanceOf(*person*,nurse)))))
```

which states that a principal has the right to book an operating room if he/she is a nurse is associated with a function `OR/book`. Therefore, if all the policy's predicates in the above policy (`instanceOf(*person*,nurse)`) evaluate to true, then the principal or subject in the nurse role is authorized to use this capability, made available within the user-specific service's binding environment. Every policy permission needs to be associated to a specific functional capability in the system (figure 6.3).

We have explored different alternatives to capture these associations between policy rights and capabilities, which can be categorized as implicit or explicit associations. **Explicit associations** between policies and system capabilities are one-to-one or one-to-many mappings

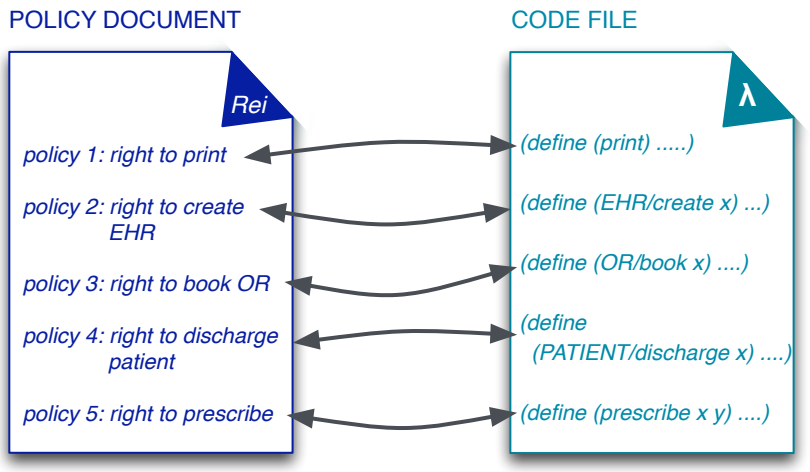

Figure 6.3: Associations between policies and capabilities.

of policy rights and function names, associations stored, for example, in a database (figure 6.4). A one-to-one relationship is for example a right to print a document with a function `print`. A one-to-many relationship is, for example, a right to access an EHR with both the functions `EHR/read` and `EHR/edit`.

Implicit associations can be made by giving both the policy action and the capability—such as a function or a binding environment—the same name. For example, the policy action *EHR/create* is implicitly associated to the function `EHR/create()` by name, and therefore this function is made available to the user. In another example, a policy action *EHR/-MANAGE* is bound to the `EHR/MANAGE` binding environment, thus making available all the bindings within that environment (e.g., `EHR/create`, `EHR/edit`, `EHR/delete`). Therefore, there is not an explicit association existing within a database, but a system's implementation needs to link policy actions and functions by name at runtime. In other words, when retrieving the capabilities that are granted for a specific principal, the system first looks for a function or binding environment in the global binding environment whose name matches the policy right. If this is not found, then the system proceeds to look up, within the explicit associations database table, the corresponding associated capabilities to be included in the user-specific binding environment.

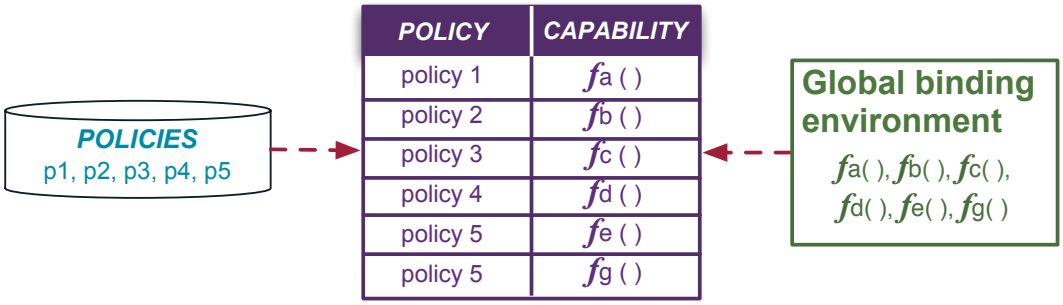

Figure 6.4: Explicit associations of policies and capabilities.

6.1.4 Service CURLs

As described in section 5.1, a CURL is a cryptographic structure that provides authority to communicate with the computation it names. In the context of our work, consumers access, use, and communicate with specific COAST-enabled services through provider-issued CURLs. A message sent through a CURL may contain primitive values, data structures, or custom closures that leverage the user-specific authorized bindings. The interpretation of this message is up to the provider.

A CURL holds different pieces of information: the issuing island, a path, the CURL's unique identifier, an expiration date (the span of validity of the CURL), the use count (may be only used n times), arbitrary metadata, and a cryptographic signature certifying the CURL's provenance, and thus its validity. For instance, a user can continue using the service as long as a CURL is not expired, has a use count greater than zero, and has not been revoked by the service provider. More conceptual and technical details on CURLs can be found in [123].

To obtain a service CURL for using an island's authorized capabilities, it is necessary to authenticate a service consumer's digital credentials. These credentials are such as unique identifiers (UUID), public keys, or any other authentication mechanism such as those based on certificate authorities or on a web of trust. Our work makes the assumption that these mechanisms are in place and that users are trusted to be who they claim they are. Authen-

tication is, therefore, out of the scope of our investigation.

Based on the user's authenticated credentials, a CURL generation service retrieves from a policy database all the policies that are relevant to the principal based on role or identity. These policies are evaluated to return a list of *true*, *false*, and *'not-evaluated* values. The only variable value that is provided, along with policies to be evaluated, for this stage of the evaluation process (service CURL creation stage) is the user's unique identifier. In other words, initially only role- or identity-based policies are evaluated; we assume that environmental conditions as well as well as conditions that depend on the target of the action are service request-specific. For example, a condition that states that a given capability is only to be used between 9:00 A.M. and 5:00 P.M. will evaluate to either *true* or *false* depending on the time of day the service is accessed.

Following, the list of capabilities associated with those policies that either evaluated to *true* or to *'not-evaluated* are retrieved. Here, we exploit a CURL's ability to embed arbitrary metadata by including information on these authorized capabilities. Since CURLs are tamper-proof, it is not possible for malicious parties to expand their authority to a provider's services by altering the authorized capability information contained in the metadata. In addition, information about the relevant policies that were not evaluated is also included in the CURL's metadata so that they can be re-evaluated at service-use time. Figure 6.5 provides a graphic description of a CURL's contents, including the metadata we include to operationalize this policy-based access control technique.

Figure 6.5: A CURL's anatomy.

The activity diagram in figure 6.6 provides a graphic description on the set of activities and decisions involved for a consumer to obtain a user-specific service CURL.

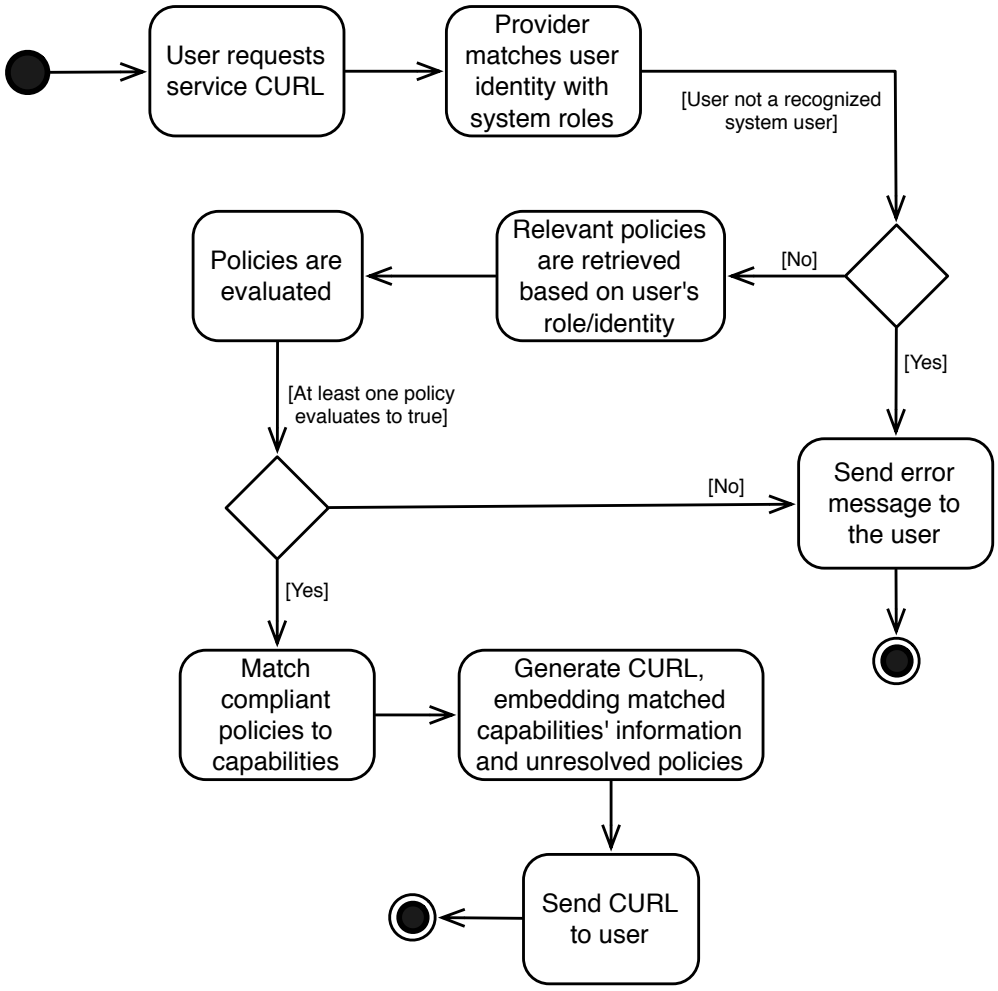

Figure 6.6: Obtaining a user-specific CURL.

6.1.5 Services Definition

An organization's services are the externalization, in a form amenable to service consumers, of information and computation capabilities (i.e. software functions). Every COAST host or *island* is initialized with a set of capabilities; for instance, a healthcare provider's island may host a database of electronic health records along with functions to create, modify, and share them, to assign roles to employees, book operating rooms, manage medical supplies, and so on. Islands can thus host a number of computations or services with diverse capabilities.

As previously described, a service is a computation which executes in the context of a binding

environment, a complex structure which holds a set of key/value pairs. This structure contains the set of primitive values, data structures, and functions (which may have access to domain data) that constitute the service's lexical scope (namely the bindings that are reachable by the service's closure) and which, therefore, are made available to consumers. In addition, environments can be merged, combined, constrained (removal of bindings), and augmented (addition of individual bindings).

In our work, we exploit COAST's binding environment sculpting in order to enable organizations to expose the desired functional capability and data as services. Organizations can compose diverse binding environments based on different binding combinations, therefore producing services that are appropriate for particular users and uses.

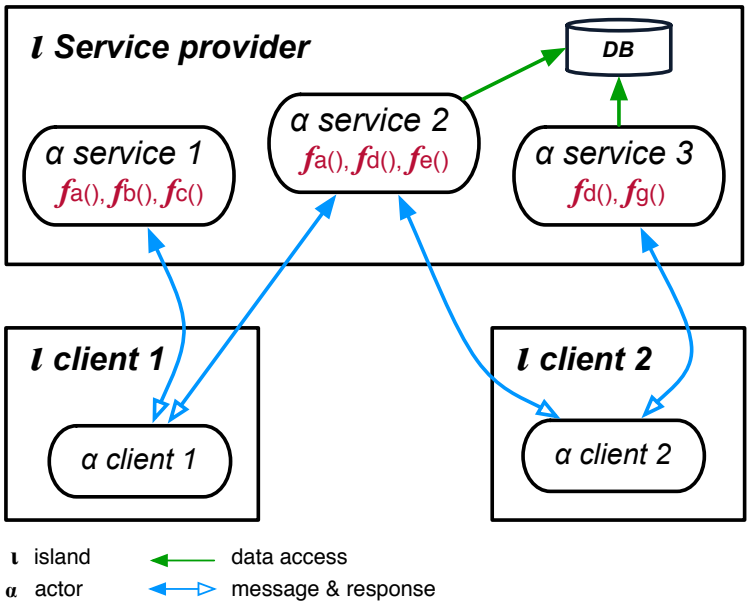

Figure 6.7: A provider's services.

For example, the service provider in figure 6.7, makes available capabilities $f_a()$, $f_b()$, $f_c()$, $f_d()$, $f_e()$, and $f_g()$. The services offered to different users are composed as subsets of this set of functions. For example, service 1 includes bindings $f_a()$, $f_b()$, and $f_c()$; service 2 includes bindings $f_a()$, $f_d()$, and $f_e()$, and; service 3 includes bindings $f_d()$ and $f_g()$. Figure 6.8 provides a more descriptive, domain specific example, where a healthcare provider offers different EHR-related services to different users: patients can access the `get-patient-ehr()` function; physicians can retrieve a patient EHR through `get-patient-ehr()` or multiple EHRs through `get-all-patients-ehr()`; authorized researchers may obtain only anonymized EHRs (`get-all-anonymous-ehr()`), and; internal policy makers may create and evaluate

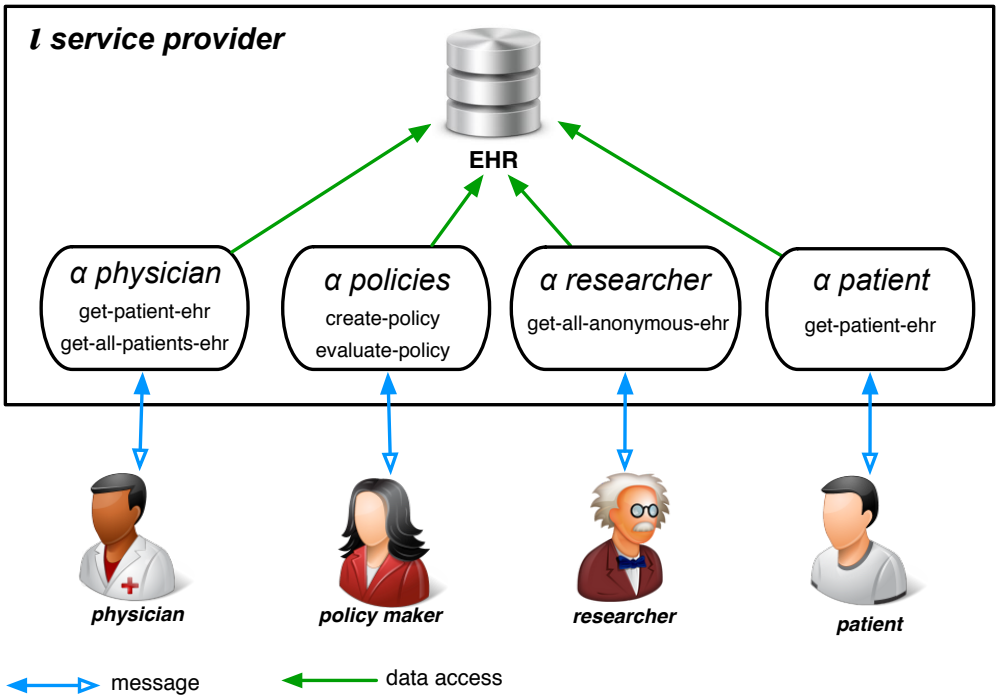

Figure 6.8: A healthcare provider's services.

policies through `create-policy()` and `evaluate-policy()`.

In sum, different services can be derived from the combination and composition of these assets—combination involves providing different permutations of these assets as services, while composition involves leveraging those assets to compose more complex ones. Composition is a key concept for customizing services, which we will address in section 6.2.

6.1.6 Dynamic Creation of Consumer-specific Services

After a consumer application has obtained a service CURL, it is ready to access the provider's authorized capabilities. As a result of exploring different mechanisms to provide user-specific services, we have identified two main techniques for creating user services. The first one is to create *persistent* services, where services remain "alive" after their creation, this being at the first service call or at an arbitrary time determined by the parent computation. In other

words, the service is constantly running, waiting for messages to process. To create persistent services, the authorized capabilities—identified as a consequence of policy evaluation—are immediately used as a binding environment to instantiate a new user service. The CURL to this long-running service is then sent to the user. In this case, it is not necessary to include authorized capabilities information—namely binding environment information—in the CURL's metadata since the CURL directly addresses the user-specific service.

A second technique is to dynamically create services which are *spawned* to serve an incoming user's request and terminate after serving it (i.e. executing or rejecting the message sent along with the CURL). The use of these services that serve and terminate is recommended when there is high variability in the frequency of service access, mostly according to diverse user types. For example, physicians may be frequently accessing EHR services, while patients will access it sporadically. Therefore, in many cases a user service may be idle most of the time. Given an exponential number of users, constantly executing user services misuse resources (i.e. memory and processing power). A CURL issued to a service user thus does not address the user-service per se, but addresses an intermediary. This proxy computation, using the incoming CURL's metadata, dynamically *spawns* a new user-specific service to process the incoming message. In more detail, the process for a new service creation is as follows:

1. The user sends a message using the service CURL to a proxy computation α*create-service*.

2. α*create-service* retrieves both un-evaluated policies and capability information from the CURL's metadata.

3. α*create-service* inspects the user's message to determine whether any of the un-evaluated policies (due to lack of provided variable values at CURL creation time) is relevant to the user's message. For example, a custom user closure may include function `EHR/delete()`; a relevant policy may describe the restrictions on the capability to delete

a health record such as record's expiration date.

4. The set of relevant policies (included within the CURL's metadata) are evaluated. If any of the policies evaluates either to *false* or to *'not-evaluated* the user's message is discarded without evaluation and an error message is sent to the user.

5. If all policies evaluate to true, the α*create-service* computation uses the authorized bindings (retrieved from the CURL's metadata) and dynamically creates within an independent thread of execution a new user-specific service; its lexical scope is composed and bound by those authorized bindings, namely, the user has only those capabilities available and nothing else.

6. Following, the user's message is forwarded to this newly created, personalized service for evaluation. The message is semantically interpreted by this computation.

7. If a custom closure exist within the message, it will only be executed if it is composed by or includes only the authorized bindings. Otherwise, the message is rejected.

8. The user-specific service terminates after the user's message has been processed (i.e. the thread of execution is terminated).

Figures 6.9 and 6.10 graphically describe this process. The sequence diagram in figure 6.9 provides an account of the sequence of steps required for a user to access a provider's services and the computations involved in the process. The activity diagram in figure 6.10 presents in detail the decision process that leads to either executing or rejecting a user's message.

There may be instances where having long running services—as opposed to services that serve and terminate—is desired. Consider for example an application where a physician spawns a long running computation at a clinical laboratory which notifies her of her patients' test results. Thus an alternative that would satisfy the constraints of our design is to provide authorized users to spawn a long running computation (service) which may have an

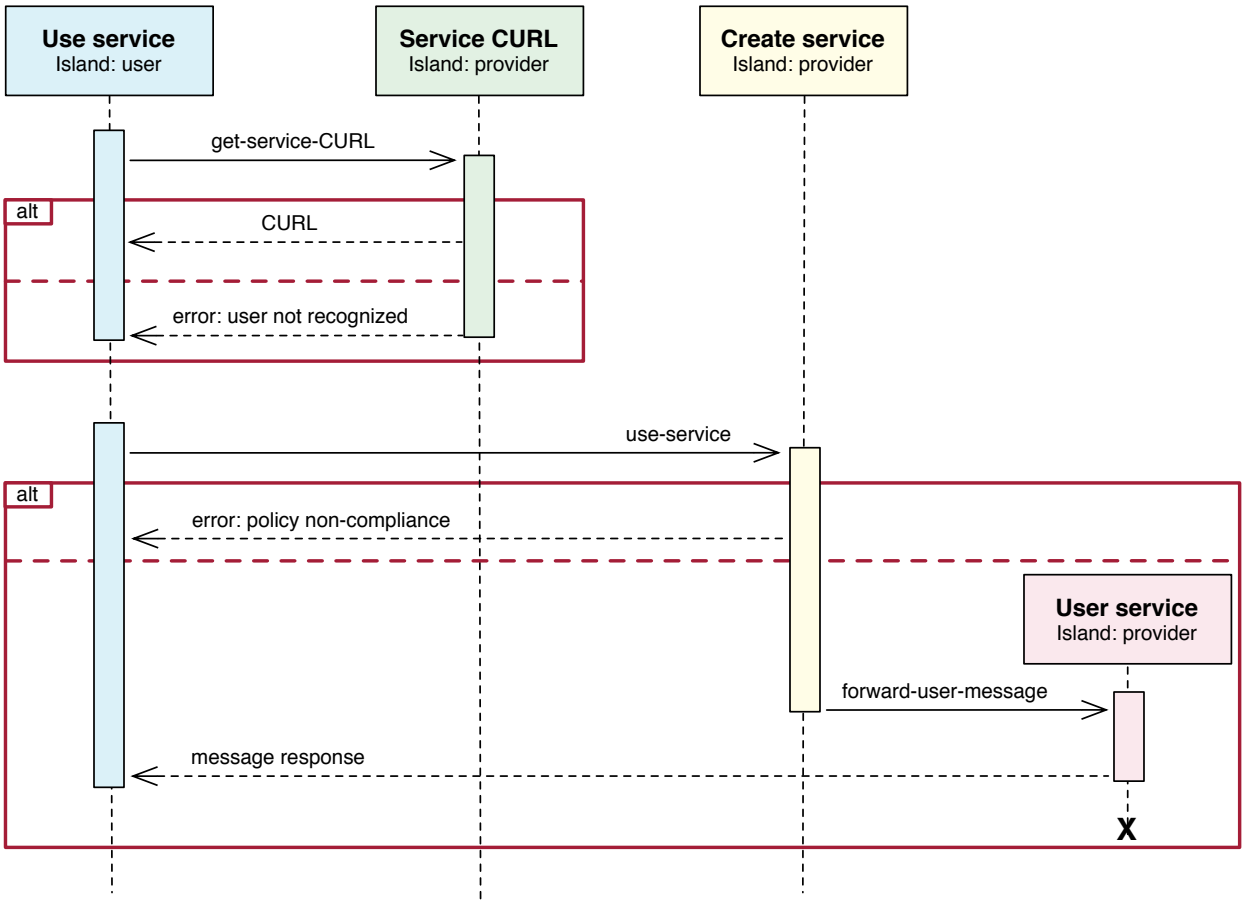

Figure 6.9: Sequence of messages for CURL request, and service creation and use.

expiration date after which it terminates. Therefore, according to the previously described steps, the user service creates this long running computation and sends the computation's CURL to the user before terminating.

Policy evaluation at service creation

As previously discussed, evaluating policies requires providing values to policy variables and executing individual policy conditions to obtain their collective truth values. We also mentioned that there is a two-stage policy evaluation process: at CURL creation time and before dynamic service creation. While some policies are based on rather static conditions (conditions which infrequently change) such as a user's role within an organization, other

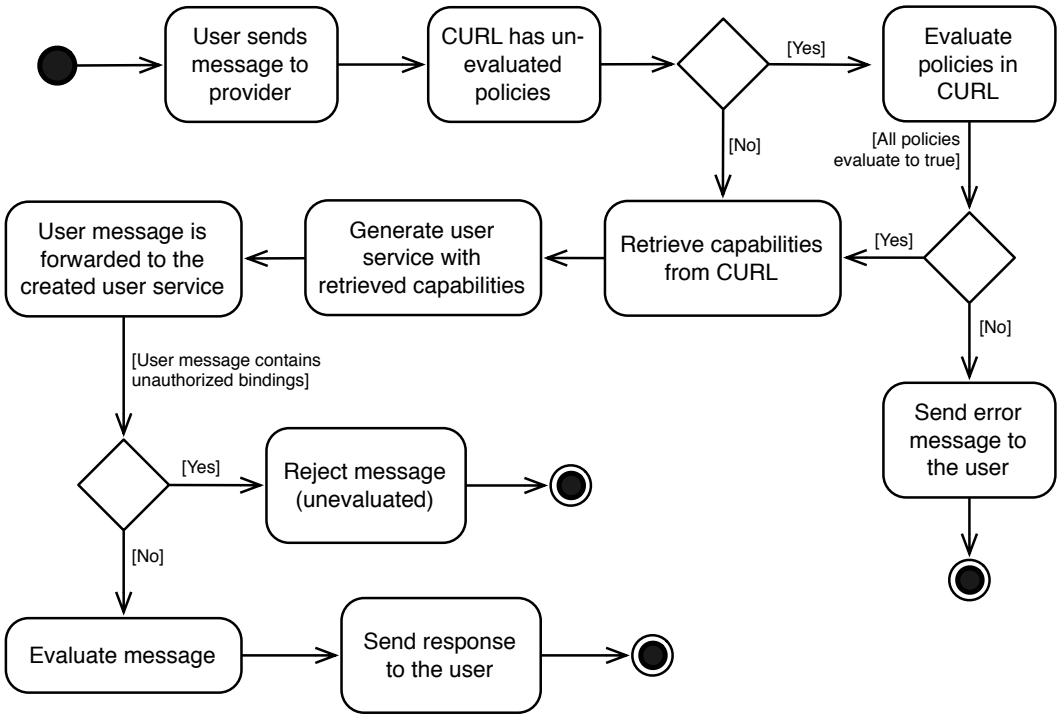

Figure 6.10: Dynamic creation of user services.

policies include more dynamic conditions such as the time of day or the day of the week to restrict access to capabilities. Therefore, at CURL creation time, known policy variable values are those related to the subject's identity or affiliation, assuming the subject is a recognized system user. So the abstract policy

```
policy(physician,readEHR,patient,null,(inDepartment(*physician*,
   traumatology), name(*physician*, John)))
```

is replaced at the first stage of evaluation by, for example,

```
policy(physician,readEHR,patient,null,(inDepartment(12345,
   traumatology), name(12345, John))))
```

therefore replacing the subject variable within conditions with the subject's unique identifier. Following this replacement, policy conditions, which are implemented as predicates with access to backend domain databases, are evaluated (thus executed) to obtain the policy's truth value.

For a policy for which at least one condition cannot be evaluated for the lack of variable values, the 'not-evaluated value (as opposed to true or false) is returned. For example, in:

```
policy(physician,readEHR,patient,null,(inDepartment(12345,
    traumatology), name(12345, John), primaryCarePhysician(*patient
    *, 12345)))
```

the variable *patient* is not provided with a value because the given value will depend on whose EHR the physician is trying to access in a service use instance and on whether the physician is the primary care doctor for that specific patient. These initially unresolved policies are, therefore, embedded within the service CURL and are left to be re-evaluated at service-creation time.

6.1.7 Capability Accounting

Keeping track of when and how a capability is being used may be important in order to restrict or terminate a relationship with a service user that is no longer considered trustworthy or to control that the bestowed capabilities are not being misused. Through capability accounting, the provider is thus aware of the capabilities apportioned to each party. Our approach supports two types of capability accounting: (a) CURL-issuing tracking, and; (b) capability use monitoring.

To oversee the issued CURLs, providers are required to keep some state, in specific the issued CURL's unique identifier, the bestowed capabilities, the island address as specified in the user's reply-to CURL (the address to whom the issued service CURL is being sent to), and a boolean value indicating whether the CURL has been revoked. A specialized computation, call this α*service-curls*, automatically stores this information at CURL-creation time (after appropriate policy evaluation). A provider may choose to only store CURLs issued to out-of-island computations to reduce overhead. This record keeping also allows the

provider to revoke the validity of a CURL, and thus revoke the given capabilities, when a user becomes untrustworthy or simply when the business relationship ends. Therefore, an incoming message is only processed if, based on the record keeper, the CURL has not been revoked. Figure 6.11 graphically describes service access in both valid and revoked CURL scenarios. The left side of the image shows a α*create-service* computation spawning a new user service to forward the user's message to after checking the validity of the used CURL. The right side of the image shows an alternative scenario where the user attempts to use the service, however the α*create-service* computation after consulting with the CURL registry, determines that the used CURL has been revoked and therefore an error message is sent back to the user.

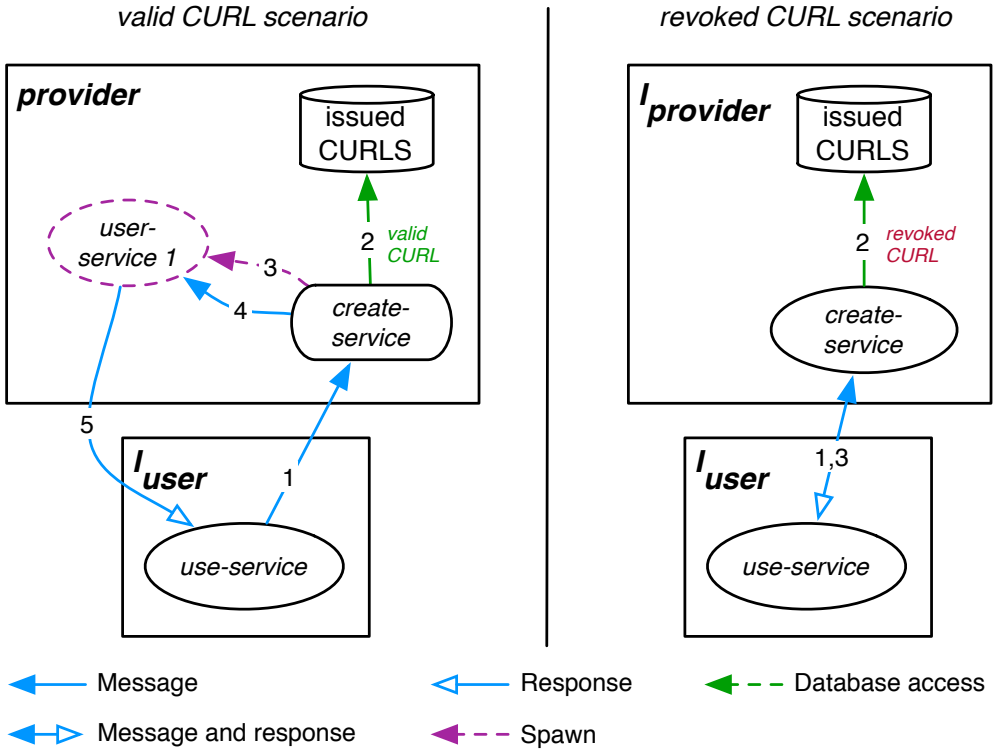

Figure 6.11: Valid and revoked CURL scenarios.

A second capability accounting practice involves monitoring the use of authorized capabilities by inspecting the types of messages that are being processed by services. A provider's log

stores both executed and rejected closures (sent in incoming messages), along with the used CURL, a timestamp, whether the closure was executed, and if appropriate the reason for rejection. A closure—and therefore the user's message—is rejected either because the policies stored in the CURL evaluate to *false* or to *'not-evaluated*, or because the closure includes bindings not existing in the authorized binding environment (e.g., attempting to make a call on `create-EHR` when that capability has not been provided to the user). Although the semantics of a closure are very difficult to analyze in an automated way, a GUI can assist a human auditor to manually inspect the code within incoming messages for potential misuse of the granted capabilities. To facilitate this manual process, message auditing can be tailored to specific needs through filtering techniques (for storage or retrieval of closures), for example, based on the criticality and the potential for misuse of the used capabilities, or on the identity service user (a provider may be interested in monitoring service use of specific customers).

Other related work

Other work has also addressed concerns related to capability accounting. For example, the PicketLink project, which addresses identity management, includes a Token Registry which stores and enables tracking an identity provider's issued and revoked security tokens [214]. Another line of research is web site use tracking [26][21][185]. Existing tools in this category include Google Analytics [120], Kissmetrics [4], and trak.io [9]. However their work addresses different concerns from our focus on provider asset privacy and security, focusing instead on using the tracked information for marketing and usability improvements.

6.2 Consumer-controlled Service Customization

Customization is increasingly desirable and important in many domains. For instance, companies have changed their business models to transition from *mass production* to *mass customization*, enabling customers to customize products according to their own needs instead of expecting customers to adapt to pre-defined product features [160]. Customization is, thus, not merely a luxury or a convenience feature within a software system. In the healthcare domain, for instance, customization is a fundamental need [193] where lack of customizability of EHR systems is a significant barrier for their adoption [57]. For example, healthcare professionals have expressed the need of customization of EHR systems for documentation activities that provide an alternative to rigid, traditional narrative charting, and therefore that account for the users' workflow [115]. Although [115] is mostly concerned with user interface customization—authors propose the implementation of customized templates—this study shows how system users, even within the same organization and within a narrow domain area (e.g., patient visit documentation), have individual preferences.

Given these motivations, our goal is to enable custom service composition through computation mobility—inbound mobile code is executed by the service provider, leveraging the available local computational and information assets. We leverage, to this end, COAST's implementation infrastructure, which enables customizing services "out-of-the-box". In the next two sections, we will describe in more detail how users can customize single-source services as well as compose more complex, multi-source services by leveraging the capabilities offered by various, decentralized and disjoint services.

6.2.1 Customizing Single-Source Services

Providers make available to users a set of capabilities which can be used "as is". In other words, COAST services allow using specific system functions (i.e. procedure call) in a comparable way to current WS* services. Therefore, a physician is capable of obtaining all patient records through a function `get-all-patients-EHR`. However, it is often necessary to post-process the obtained information for specific purposes, a common challenge we described in 6.2. For example, a physician may only be interested in finding out what is the average age of patients with diabetes. In this scenario, obtaining all patients records is a misuse of bandwidth and local processing power. These are the kind of scenarios where the power of customization is very valuable.

Customization can be achieved through functional composition—the provider offers a set of authorized capabilities which can be used by service users to compose new services suited to their specific needs. In other words, given that actors themselves are computational environments, the available bindings that compose the lexical scope of an actor can be combined to compose custom closures. A provider, for example, may offer capabilities fa, fb, and fc (figure 6.12). A user computation may send a message with the closure `(lambda () (c (a x)))`, which is executed by a provider's service to obtain some result.

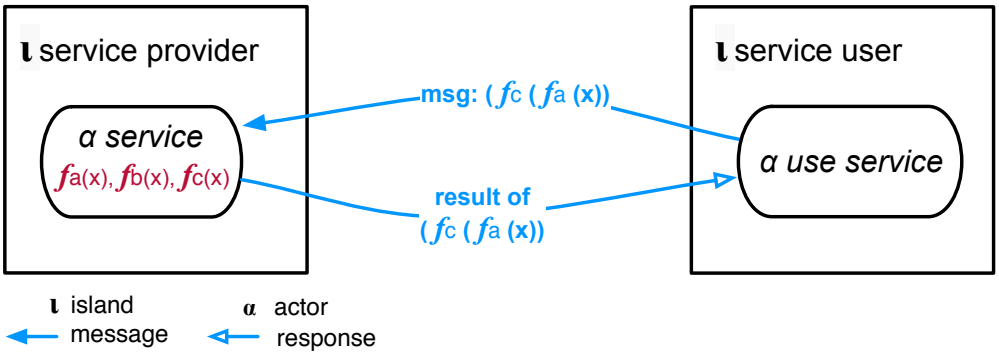

Figure 6.12: Service customization.

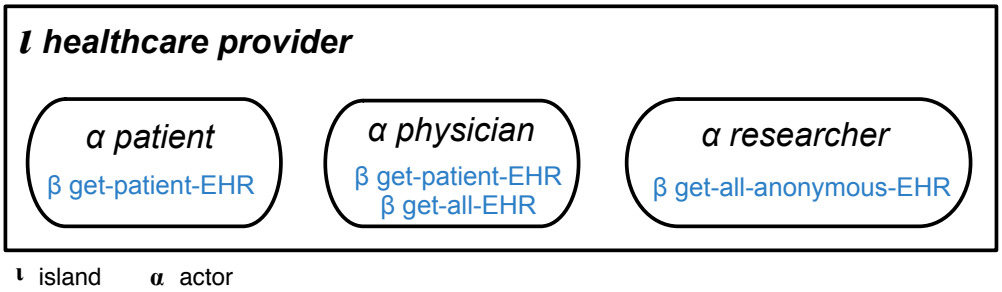

Figure 6.13: A COAST peer running 3 user services.

To provide a more concrete example, a healthcare provider's COAST peer may have, at startup, three services running (figure 6.13): one for John ($\alpha patient$), another for Dr. Smith ($\alpha physician$), and a third one for Dr. Jones ($\alpha researcher$), each with their corresponding binding environment (denoted by β) [1]. Dr. Jones, using a CURL *@researcher* sends $\alpha researcher$, for example, a custom closure $\lambda average\text{-}age$ computes the average age of patients with diabetes by leveraging the provided `get-all-anonymous-ehr` domain function (figure 6.14). The output of function `get-all-anonymous-ehr` is a vector of hash tables, where a hash table contains an individual patient's information. Although the function to obtain the desired data was not explicitly provided by the service, COAST allows customizing services by way of function composition and code mobility. An example of such custom service composition is:

```
(define average-age
    (lambda()
        (let* ((patients-vector (get-all-anonymous-ehr))
               (num-of-patients (vector/length patients-vector))
               (vector-of-ages (vector/map patients-vector
                                 (lambda(x) (hash/ref x 'age #f)))))
          (/ (vector/fold/right vector-of-ages + 0)
             num-of-patients))))
```

[1] Figure 6.13 illustrates a partial view of the system, depicting only the service provider.

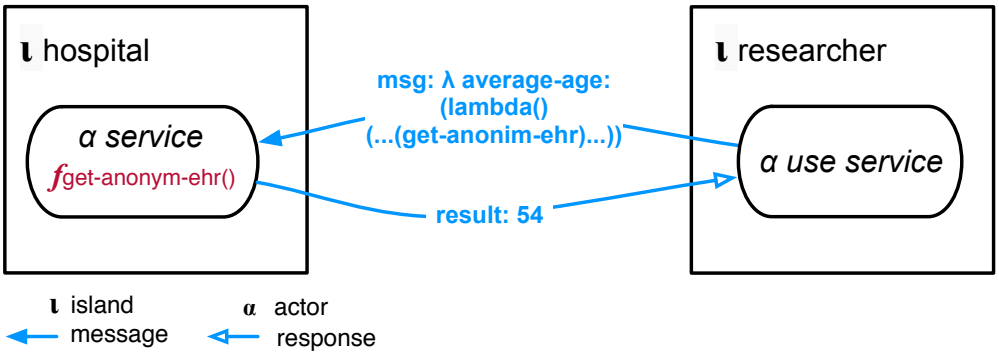

Figure 6.14: EHR service customization.

Also, as described in 6.1.6, a custom closure will execute only if it is composed using the authorized bindings. For example, a user service may provide capabilities `make-ice-cream`, `add-nuts`, and `add-fudge` to create custom ice-cream images. A custom closure `(λ () (add-fudge (add-whipped-cream (add-nuts (make-ice-cream "chocolate")))))` is discarded, since the capability to `add-whipped-cream` has not been provided to the user (i.e. it is not available in the service's bounded execution context, even if the function exists within the island). Services or computations are closed execution environments where the user is constrained to use only the authorized functions and data made available to him/her. Thus, enabling appropriate differential access demands fine-grained management of these functions to curtail or augment composition power and access to data.

As described by these examples, our approach and accompanying platform provide security of information and capabilities, yet the expressiveness of functional composition. The foundations of the COAST infrastructure on the lambda calculus allow this expressive power. The lambda calculus is a universal model of computation where functions can be provided as arguments to other functions, and where functions can be combined to compose other operators [135]. Not only it is possible to construct these fist-class functions (functions which take functions as arguments), but also higher-order functions (functions which produce other functions) [31]. Given these powerful formal, mathematical foundations [72], COAST-based services provide a degree of flexibility to customize services that cannot be provided with-

out mobile code and functional composition; the provided functions are used by service consumers as the building blocks of custom-made services.

6.2.2 Customizing Multi-Source Services

Not only is it possible to compose custom services from a single provider, but computation mobility allows processing and integrating data from multiple sources. Just like current mashup technologies, our aim is to integrate decentralized components and information into a single application. However, the power of code mobility provides us with increased flexibility and expressiveness to compose services and applications with decentralized constituents.

Custom services can be composed with multiple services' capabilities controlled by a single or by multiple providers (i.e. exist within a single or multiple decentralized islands). The way these custom services are composed are specific to the consumer application's purpose and information needs. We identify three patterns of message exchange among computations to achieve custom service composition, patterns which can be associated to well-known network topologies [250]: (a) in the "independent communication" star pattern, a user performs independent remote computations from different intra- or inter-island services independently, retrieving information and locally processing a combined result from the set of individual service requests 6.15; (b) in the "dependent communication" star pattern, the user application invokes services in sequence, using the result of one service as the argument for the next one 6.16, and; (c) in the ring pattern, custom closures travel through multiple services, processing messages and producing intermediate results that are sent to the next computation before returning an aggregated result to the user (figure 6.17). Although figures 6.15, 6.16, and 6.17 depicts services within different islands, this typology applies as well if they existed within a single island, thus managed by the same authority. Under this type of classification, single-source service customization described in 6.2.1 follows a point-to-point topology.

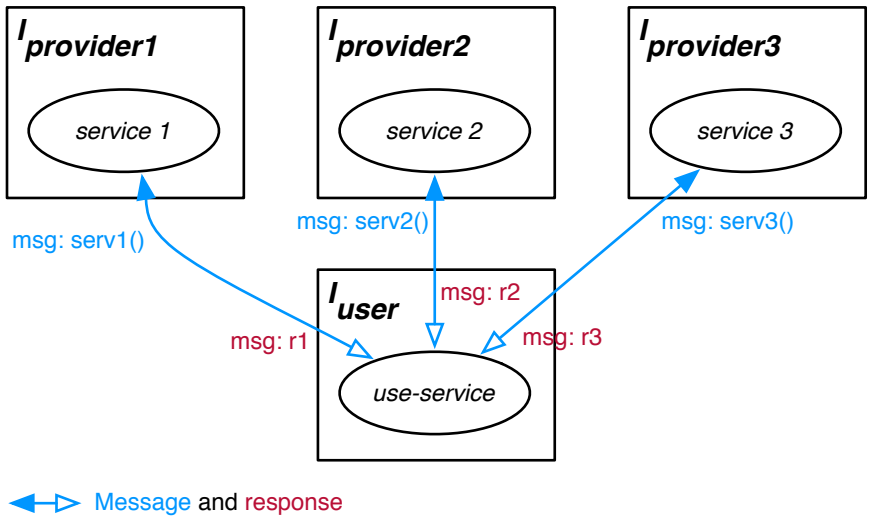

Figure 6.15: Independent star topology.

Figures 6.18, 6.19, and 6.20 illustrate scenarios based on these service composition patterns. Figure 6.18 describes a scenario where Dr. Jones, the researcher from the example in section 6.2.1, is interested in comparing the average ages of diabetes patients by state to further investigate the reason for their variation (if any). He sends independent custom closures to different health organizations across states to retrieve this information. All service requests are disjoint and sent independently of each other (independent communication star).

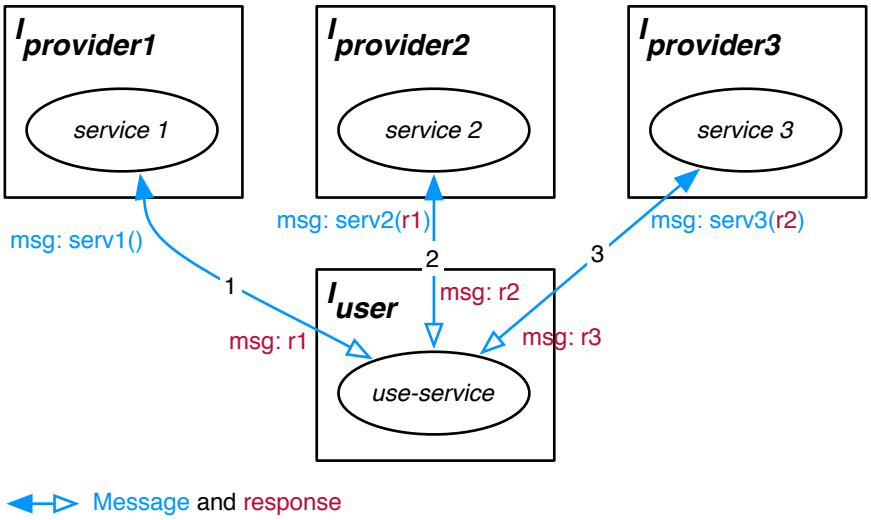

Figure 6.16: Dependent star topology.

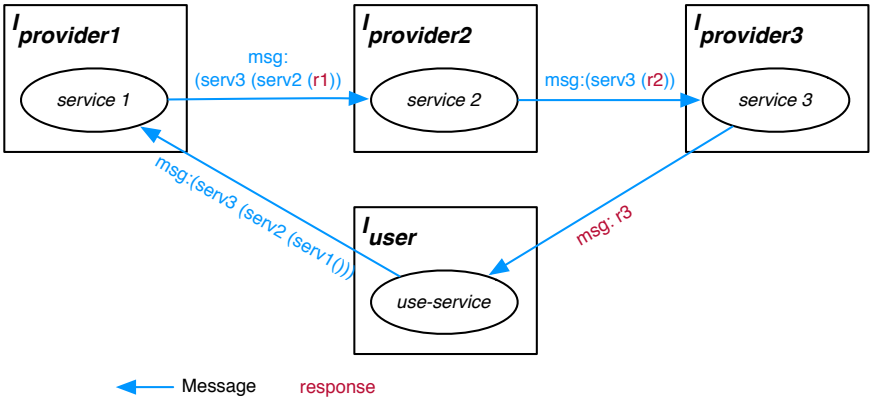

Figure 6.17: Ring topology.

Figure 6.19 instead depicts a patient who places a prescription order through a pharmacy's online service. He first retrieves the prescription from his EHR at the health center and then uses that information to place the prescription order. Numbers on top of the arrows indicate the order of events. Therefore, the result retrieved from one service is used as an argument for another's invocation (dependent communication star).

The scenario in figure 6.20 is a more complex and interesting one; Ann, a hospital admin-

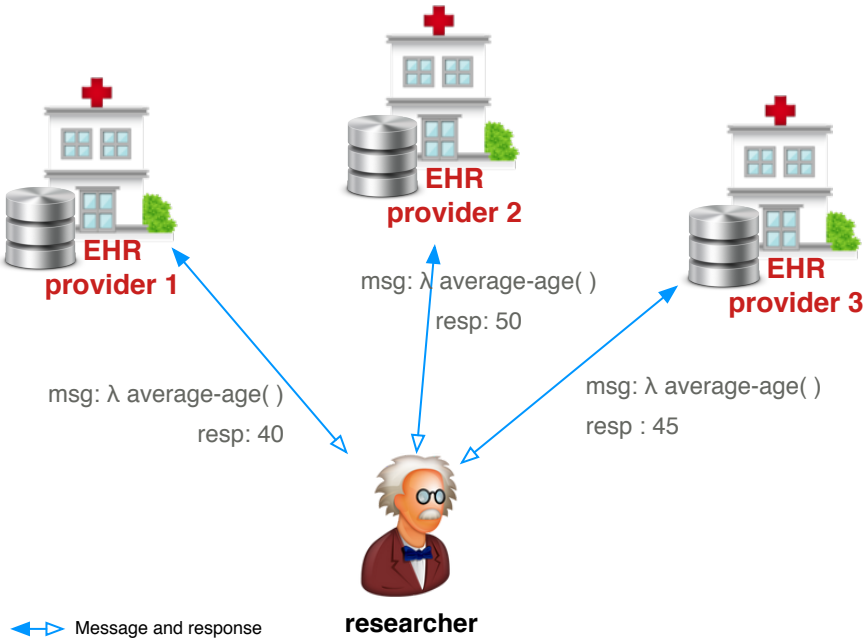

Figure 6.18: Independent star topology scenario.

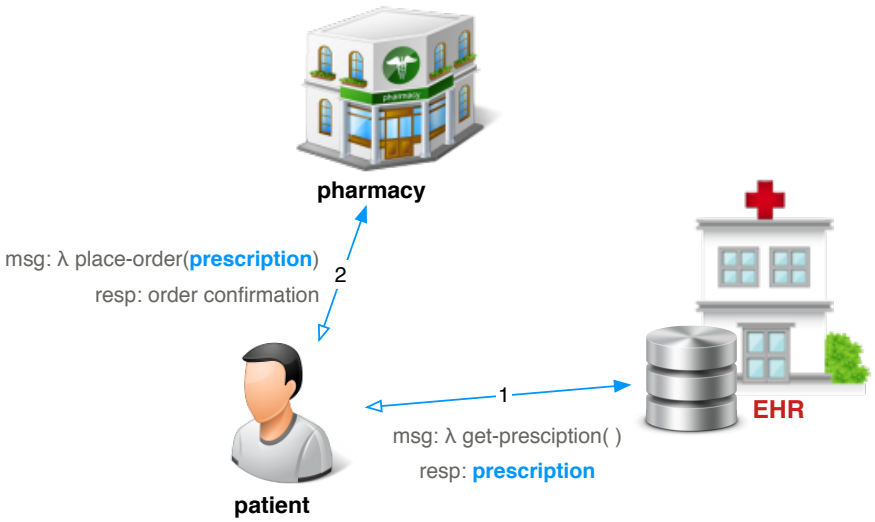

Figure 6.19: Dependent star topology scenario.

istrator, is in charge of the drugs inventory within a hospital. Using an application in her PDA, she sends a custom closure to a computation which handles the EHR database to calculate the weekly needs of medication based on the consumption rate of a set of drugs. The result of this computation is then forwarded to a hospital service which manages the drug inventory. The custom computation compares the weekly needs with the current stock, therefore calculating the amount of each prescription drug to purchase. Ann's custom computation continues its path towards the pharmacy's COAST island to place an order given the needed drug quantities. Finally, a confirmation of the placed order is sent back to Ann.

The following code snippet is an example of such custom closure. This last scenario truly distinguishes COAST-based services from current services technologies. Enhanced expressiveness through custom, mobile closures enable richer multi-source service compositions that satisfy a wider audience of users.

```
(curl/send @EHR
  (λ()
    (let ((required-drugs (calculate-drug-stock-needs
                           (get-stock-info)
                           (weekly-drug-consumption-rate))))
    (curl/send @pharmacy (λ()(place-order required-drugs))))))
```

6.2.3 Comparison to Other Related Technologies

Other studies have also addressed customization. Mashups, for instance, emerge from the need of application composition and customization leveraging remote services. Mashups' innovation was departing from developer-controlled service composition to enable consumers—mostly novice developers—to create their own applications from existing web services [173]. Liu et al. [173], for example, propose a "mashup component model" which provides a set of

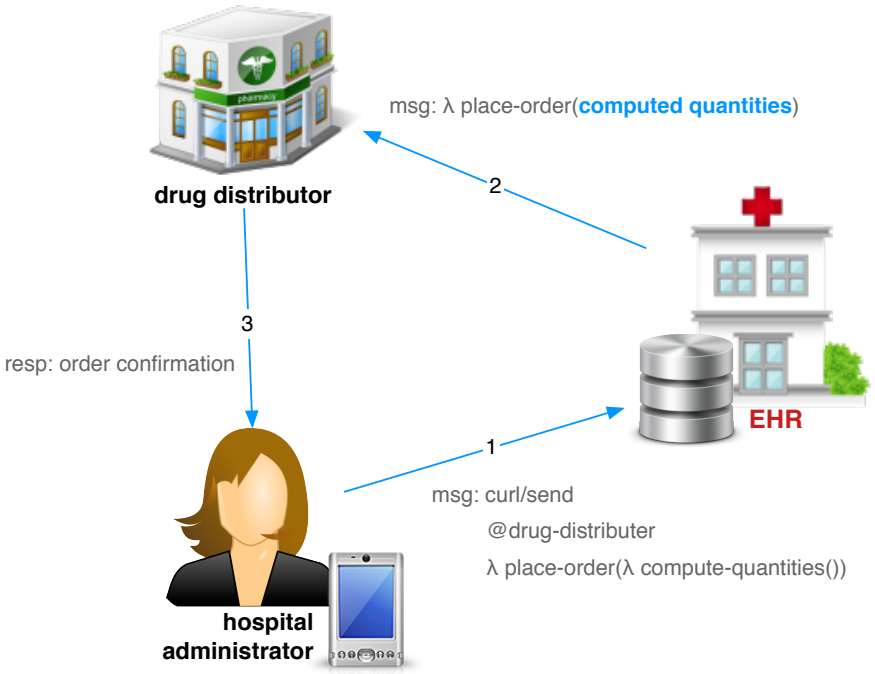

Figure 6.20: Ring topology scenario.

UI components that are linked to backend services selected from a service catalogue such as UDDI. End-users can then drag-and-drop in the browser these visually represented services to compose custom ones. However, as the author argues, mashups are lightweight and end-user oriented composition technologies which are not suited to the types of data-sensitive, privacy-aware, and large-scale applications that we address in this research. In addition, Liu's approach still requires semantic and technical interoperation of services backing this visual service composition technology. Composing mashup applications involves "significant manual programming effort" [274]. Based on our first-hand experience building a mashup involving a travel and leisure guide during a course project, composing new applications using existing services as building blocks is difficult at best. Semantically and structurally different, and often incompatible services, as well as the unavailability of needed services, brings about systems which misrepresent their initial design. In addition, often times most of the data returned by the service is useless for the client, thus discarded, misusing bandwidth and processing resources in the process of service composition.

SOA technologies also contribute to service composition and business processes through technologies such as BPML, XLANG, WSFL, WSCL, BPSS, and BPEL4WS. However, and as Lui et al. and Mandell et al. [181] point out, these are complex technologies that have become barriers to service composition and have not achieved the goal of seamless interoperability. In addition, a significant shortcoming of current web services technologies is the lack of consideration for the needs of a diversity of users—"one-size-fits-all" services constrain users to work around them to obtain the desired information. The current norm is that users need to adapt to services as opposed to services adapting and enabling users to satisfy specific needs.

Other research efforts, leveraging SOA technologies, have approached service customization [181][61][228][245]. Mandell et al. [181], for example, leverage semantic web technology to augment the BPEL4WS process modeling technology with a Semantic Discovery Service to

enable custom service discovery and composition and improve service interoperability. In this context, customization refers to the ability to automatically select constituent services that meet the constraints of individual users (the approach excludes automated composition and the service workflow is fixed). Cao et al. [61] leverage as well domain ontologies to address the dynamic definition of service process models according to customer requirements. Although users can find services that meet specific constraints for composition, individual services used as building blocks are rigid, non-customizable solutions, therefore more limiting in terms of composition capability. In addition, and according to Liu et al., once these SOA-based composite services are deployed, it is difficult to customize them according to changing users' needs given their long development lifecycle.

In another study, Nguyen et al. [201] leverage product line techniques to model service variability, thus enabling users to tailor services to their needs. Sam et al. [228] go a step further to allow the automated transformation of web services' configurations, published in directories, to others more suited to consumers' needs. Stollberg et al. [245] similarly approach service variability and complementary modeling techniques for managing such variabilities. This research however, involves configuration of service interfaces rather than customization, where the service provider defines the allowed service variations. Our notion of customization is far more flexible, expressive, and more user-controlled than what Nguyen et al. propose, and not limited to one, but to multiple decentralized services. In addition, this line of research focuses on service inputs and output types, however not on manipulating the core functionality and behavior of the service enabled by COAST's mechanisms.

6.3 Limitations and Scope of this Work

Despite the novelty and expected benefits of our approach towards service customization and differential access, there are some limitations and open questions that we do not address and are out of the scope of this dissertation proposal.

Automated service discovery. COAST is silent on automated mechanisms for discovering services offered by islands. Therefore, developers need to turn to (out of band) island- and site-specific documentation in order to compose custom services. However, a current priority and a first step is laying out strong foundations on how to effectively exploit, protect, and customize those services before focusing in automated discovery.

Trust and reputation. In decentralized systems, where private and critical data is exchanged, mutual trust among parties is essential. The challenges in this aspect are (a) determining whether the service user is who he claims he is, and (b) determining whether this authenticated party is reliable or trustworthy. While the former challenge is a matter of "identity trust", the latter deals with a party's reputation. An example of this distinction is the following: a party attempting to access a service may claim to be a government agency; although it is considered trustworthy, it is necessary to verify its identity (i.e. public key associated with a real world identity). Both identity and trustworthiness are necessary so that service CURLs are issued to the right hosts. On the other hand, a research institution's identity might have been established, but the provider may not know if this is a reliable user. In this case, it is necessary to assess, for example, its academic reputation.

Unfortunately, COAST island self-certification is not enough to promote trust among parties. More specialized trust management is necessary to dynamically assess the trustworthiness of other participating parties when trust boundaries are crossed [157]. Existing trust and reputation models can be integrated with our approach [127]. For example, the semantic web community proposes a model where users have a combined trust score based on multiple

individual scores given by users [224]. To obtain reputation information we can refer to models such as community-based models used in P2P networks [10][263], specially those classified as distributed reputation systems [146][92].

Although implementing existing trust models or creating new ones is beyond the scope of our research, we make explicit some assumptions on how parties are authenticated and how trust is promoted to carry out fruitful interactions. We assume a "Web of Trust" model where each party chooses who to trust [62]; parties have available a trust graph, such as the one described by Hamouid et al. [132], where nodes authenticate public keys of other nodes based on social relationships without relying on a centralized certificate authority. Each node has a publicly-known identifier, cryptographically bound to a private key by way of a "witness" that is generated by another node. In addition, we assume that a provider, through an appropriate communication protocol, obtains service users' reputation ratings based on recorded past experiences from other known service providers.

Privacy policy languages. Although our work involves deriving user services from specific privacy policies, our work is not focused on neither creating a new privacy policy language (given their availability) nor recommending specific languages to describe policies in our domain or in a set of domains. However, we leverage existing policy languages for our scenarios and describe the rationale of our choice. Our approach, rather, focuses on demonstrating the feasibility of deriving binding environments and CURLs according to a set of policies, independently of their specification language. That said, future work is necessary to assess whether particular languages are appropriate for specific domains, whether domain-independent languages can be used across many domains, or if new languages ought to be created due to the inability to capture the desired semantics with existing ones. Depending on the specific situation, domain specific languages can be created to allow specifying domain-specific privacy policies.

Statistical privacy techniques. Although differential access to providers' services sup-

ports individuals' privacy by allowing only authorized parties to access personal information, our approach does not solve the privacy problem, namely the unauthorized dissemination and misuse of personal information. Parties may have background information [198] that in combination with the data obtained by service providers allows making inferences to de-identify and discover private information about individuals [102]. Our concern is rather the unauthorized disclosure of specific data about individuals. This said, service providers may well implement diverse privacy mechanisms available [12] as a core part of the service to prevent disclosing personal information that is too revealing and which may compromise individuals' privacy. This work also excludes concerns, however legitimate, of how personal information obtained through a service is used; in fact, in the healthcare domain, HIPPA itself states that "there are no restrictions on the use or disclosure of de-identified health information" [96]. Although relevant to our line of research, we also exclude from the scope of this research the persistence of data provenance information across transactions and research related to "sticky" policies [192].

System usability. Although we provide basic interfaces as part of our experimental EHR system, we do not focus on specific usability issues for the notional participants of our test domain (e.g., patients and doctors). We instead provide the core mechanisms to enable service customization and differential access that are essential in order to build domain-specific tools that consider usability issues for the specific system users.

Access control for services within processes. In our work, we do not specifically consider services which are part of larger processes. The policy languages and access control mechanisms for these type of services which are part of an orchestration or a choreography of services usually include constructs for, for example, stating that action A ought to be called before action B, or that the user is obligated to perform action C every time action B is invoked. For instance, Rei itself counts with syntax to support this type of coordination semantics and action sequences. These concerns are rather relevant within workflow languages

such as YAWL [258].

Our approach allows calling services independently from each other. However, interesting future work could support workflow through coordination of services whose dependent invocation is guided by formal policies.

Chapter 7

Practical Experiments: COASTmed

To assess the practical feasibility of our approach we conduct experiments involving system development. To this end, we developed COASTmed, a prototype founded on COAST's architectural constraints and techniques and formal privacy policies. COASTmed is an EHR management system which, supporting decentralization, offers services to diverse users with distinct service access privileges. The development of this application is part of the evaluation of our approach, specifically aimed at answering the question on whether we can successfully build in practice Internet-based services which exhibit the desired properties with the proposed techniques (formulated in section 3).

The system properties that we aim to exhibit with COASTmed are: (a) provider-controlled, policy-driven differential service provision, and; (b) consumer-driven customization. In other words, COASTmed enables providers to differentiate among users who are entitled to different information and capabilities, while at the same time allowing users to flexibly customize the use of such services within the boundaries imposed by the provider. COASTmed is also the artifact for the scenario-based evaluations in section 8.1.

COASTmed can be downloaded from `https://bitbucket.org/abaquero/coastmed`.

In following sections, we describe the approached problems in the context of the healthcare domain, in specific with respect to the use and sharing of electronic health records, and motivate the choice of this experimental domain (section 7.1). In addition, we describe a set of forward looking scenarios from a COAST perspective regarding the on-the-spot access to healthcare information. Then, we provide a description of COASTmed's initial architecture (section 7.2) and a description of the leveraged domain data model (section 7.1.2). Following, we explain more in-depth how access policies for the provided services are specified (section 7.3), how user CURLs are created and issued to domain users 7.4, and how these CURLs are used to dynamically create a user service and serve a user's request (section 7.5).

7.1 Empirical Domain: EHR Management

Healthcare is largely a decentralized enterprise—a network of people and organizations are involved in some aspect of patients' healthcare, frequently accessing and sharing patient information for different purposes (figure 7.1). A nontrivial consequence of these interactions is the dispersion of patient data among uncountable, decentralized digital locations. Despite HIPAA regulations and other locally imposed privacy and operational policies to protect patients' data, there are precedents of privacy breaches due to, for example, insiders' authority abuse [225]. The preeminent problem is that these policies may be vague and often detached from the operation of information systems. Consequently, internal and external organization services do not always comply with these regulations.

A second, no less important problem is that patient information is used for diverse (often unanticipated) purposes. Yet, it is unfeasible for healthcare organizations holding patient data to deliver personalized information for every current and future user and need. Although Web service technologies are being deployed in this domain to deliver patient information in replacement of exchanging physical documents, these are rigid, unilaterally controlled

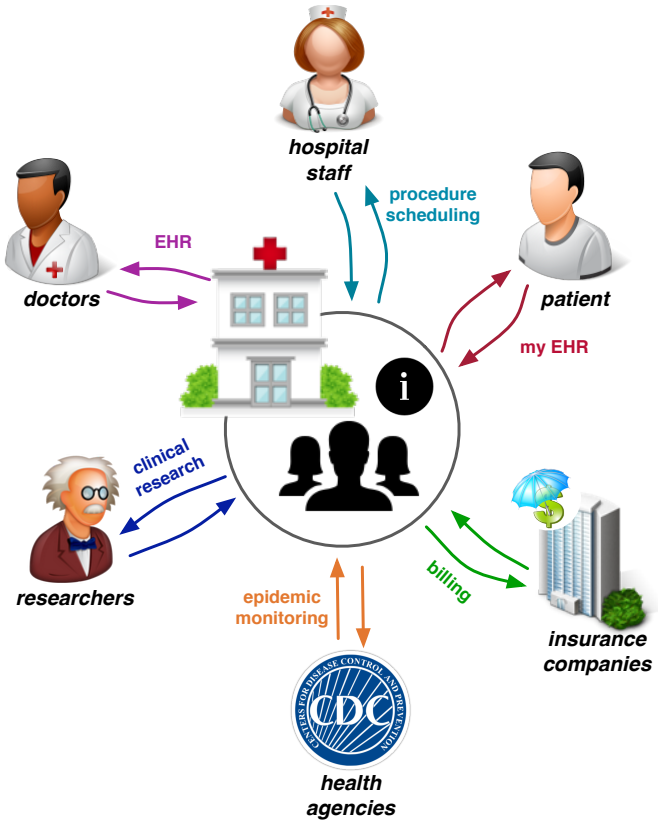

Figure 7.1: Parties and interactions in the healthcare domain.

solutions which hardly satisfy their users' needs and curtail access rights to otherwise more privileged users. As we mentioned in section 6.2, customization is a fundamental need in this domain, where users, even within the same organization, have individual information needs (regarding, for example, content, format, visualization, and interaction modes) with respect to patient data.

Given these challenges, there is a need for more suitable web services for this complex domain involving an intricate network of users and uses of patient data, heterogenous trust relationships among parties, and convoluted legal and privacy strictures with respect to management and disclosure. We identify two salient challenges for these novel information services: (a) granting access to patient data according to appropriate, desired relationships between individuals and organizations, and; (b) allowing authorized user applications to obtain the required patient data for diverse uses.

Our work approaches the essential need of speedy and safe exchange of distributed patient data in support of efficient and coordinated healthcare delivery, epidemic outbreak surveillance, health sciences research, and other socially beneficial goals. COASTmed is meant to provide customizable EHR information services to independent and heterogeneous users. The critical nature of data, the number of participating parties, and the various trust relationships in this complex and highly decentralized domain provides a fine example that leads to the exploration of both the challenges and the enabling technologies for the secure and privacy-aware collaboration between parties to share, access, and use health-related data and capabilities.

7.1.1 Rethinking the EHR Scenario: Sharing and Using Patient Data in COASTmed

The following scenarios address shortcomings in the domain regarding the lack of immediately available data, systems and services policy compliance, patient information privacy and security, the invariability and inflexibility of information services, and the integration of distributed and independently managed data.

When John's electronic record was first created, a CURL was issued to him by the healthcare provider to enable accessing his health record at any time from either his laptop or mobile phone. This CURL allows him to see only his medical record and no other information or system functionality. Doctors and nurses are also authorized to access John's and other patients' data to provide them with medical treatment. Principal-specific CURLs provide healthcare employees individual authorities with respect to the access, management, and manipulation of electronic records.

At a later time, John accesses an online service to make an appointment with Dr. Smith. At the appointment, John uses a mobile user interface assistant to send a single-use CURL to

the doctor's email to allow her to access, on the spot, a laboratory's data service to obtain John's blood work results. The issued CURL is created upon the patient's request by the lab according to its own privacy policies. Dr. Smith can only access the most recent results and the CURL cannot be used again. These restrictions are embedded in the CURL to preserve the patient's privacy. Using the issued CURL, the doctor sends a custom algorithm to the lab's service to perform specialized data analyses and diagnostics on John's health data.

Jane has been diagnosed with hip osteoarthritis and requires a hip replacement. Concerned about the expenses, Jane uses a CURL issued by her physician to access her health record where the recommended procedures are recorded. She sends a custom computation with the procedure's code to her insurance company's online service by way of a policy-holder-specific CURL. The service identifies the patient through metadata encoded within the incoming CURL. The custom computation executes, calculating the expected expenses based on her insurance plan. The insurance company also allows Jane to compare the costs of various providers in the area using a patient-specific service.

Researchers are investigating the relationship between diabetes and cardiovascular diseases. Multiple healthcare providers, according to their own privacy policies, issue a CURL for the researchers use. The capabilities conferred by CURLs restrict the access to the specifics of patients' records, but enable researchers to retrieve aggregate patient data. Given that services are computational environments, researchers may send custom closures such as custom statistical algorithms to their researcher-specific service. For example, using the data hosted by the service provider, the custom closure computes the average age of patients diagnosed with diabetes. This closure can only leverage the capabilities authorized, thus those available within lexical reach of the service.

The issued CURL has both time and use count constraints—researchers can only access the provider service at most a hundred times as defined by the provider during the validity of the CURL, namely it expires a month after it was issued. If researchers attempt to use

the issued CURL after the permitted use count has been exhausted or after the CURL's expiration date, the service request will be denied, thus the user message is discarded.

In addition, healthcare providers offer different services to a wide range of users. For example, pharmaceutical companies use services for the evaluation of prescription drugs through periodic analyses of patients' treatment and outcomes; the CDC installs long-running computations in key healthcare facilities to detect epidemic outbreaks. Healthcare providers generate and distribute CURLs addressing policy-conforming, privacy-aware, and specialized services which provide the degree and depth of access parties have agreed upon, limiting functional capability to their specific authority.

7.1.2 EHR Data Model

A data model describes, at a high level, the domain concepts and the relationships among them, and at a lower level the way these are organized, stored, and accessed within a software system. In our domain, the data model largely involves patient's health information such as symptoms, laboratory results, and diagnoses.

To obtain a realistic data model for the COASTmed prototype, we looked at various open source healthcare information systems such as PrimaCare [179], VistA [206], WorldVistA [268], AstronautVistA [24], FreeMed [111], GNUmed [3], OpenMRS [7], and OpenEMR [6]. At the heart of most of these systems is an electronic health record management component such as the Computerized Patient Record System (CPRS) in VistA. Most have very rich data models, which reflects the complexity of the domain. VistA is particularly complex, not only composed of the CPRS but also of modules for pharmacy and laboratory management, clinical imaging, dentistry, billing, a patient portal (MyHealthVet - www.myhealth.va.gov), and other specialized functions for patient treatment and hospital administration.

In order to conduct practical experiments on the interactions between the EHR management system and diverse parties, we chose to use PrimaCare's data model due to its richness and detail as well as the availability of a small sample patient data.

PrimaCare is an electronic health record management system developed by the Primary Care Doctors' Organization Malaysia (PCDOM) with the goal of improving healthcare delivery and providing a standard interface to different associated hospitals, clinics, and healthcare offices. The overarching components of PrimaCare's data model are health record management, an ontology of standardized concepts, and a set of administrative functions such as accounting, billing, and room scheduling. Health record management includes patient demographics, medical history, diagnoses, treatments, interventions, and so on. Appendix C shows a detailed list of PrimaCare's data model elements.

In addition, PrimaCare uses international healthcare standards such as the International Classification of Primary Care (ICPC) [265]—which describes patient's reported symptoms or complaints, diagnosis, treatments, interventions, test results, and referrals—the International Classification of Diseases (ICD) [208], the Anatomical Therapeutic Chemical (ATC) drug taxonomy [267], and the Logical Observation Identifiers Names and Codes (LOINC) describing laboratory and clinical observations [186].

7.2 Initial System Architecture

Electronic health record management applications are complex systems composed of tens or even hundreds of components encapsulating different aspects of the healthcare delivery and management process (e.g., EHR management, drug interaction checking, prescription refill, home monitoring, and more [91]). COASTmed focuses on the secure service provision aspect of an EHR application, thus the components of our architecture heavily address access

control mechanisms.

Figure 7.2 features the initial architecture of COASTmed, where a root computation α *root* creates, at startup, three services or computations: (a) α *service CURLs*; (b) α *policy manager*, and; (c) α *user services*. Each of these services are reified as closures executing in independent threads of computation, receiving and processing incoming messages, and each with its own capabilities or binding environment. Computations' capabilities, namely the functions within running closures' lexical scopes, are assigned at their creation time by α *root*, their parent computation, comprising a purposefully selected subset of the root's capabilities.

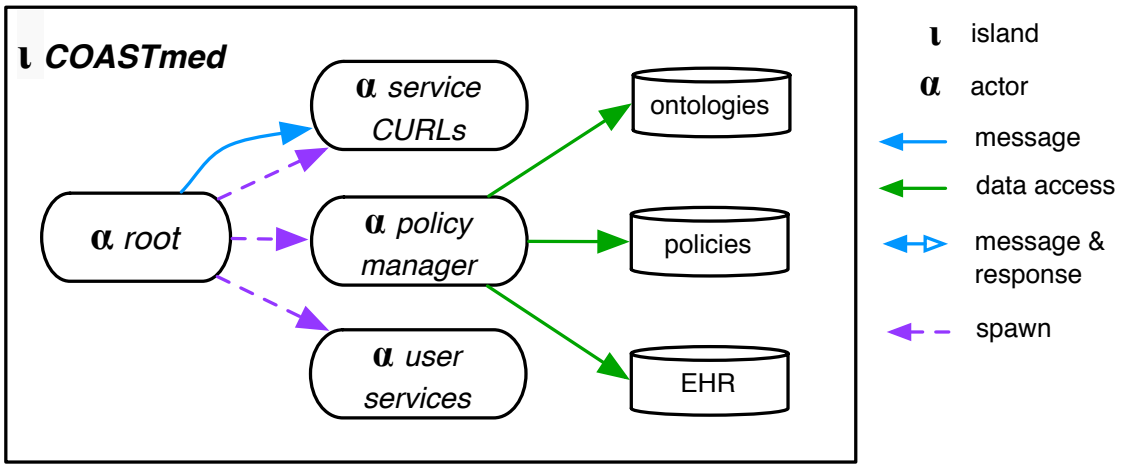

Figure 7.2: Initial system architecture.

α *service CURLs* generates user-specific CURLs that allow individual users, based on their authenticated identity, to access authorized services; the CURL stores as metadata the names of the authorized capabilities (binding environments or functions) for the user the CURL is being issued to. To obtain this CURL, users send a message to α *service CURLs*.

To create CURLs, α *service CURLs* coordinates with α *policy manager* which handles policy creation and evaluation. CURLs are created based on the result of evaluating policies that are relevant to a specific user based on identity or role within an organization. To create and evaluate policies, α *policy manager* has access to: (a) domain ontologies which define the

terminology for policy specification; (b) the policy database to record and retrieve policies, and; (c) domain information such as the organizational structure and patient's records to obtain facts, the basis for policy evaluation.

Finally, α *user services* is the computation responsible to create user-specific services in response to a user's message. The behavior of this computation reflects Agha's actor model [13] which is foundational to COAST's principles: a computation implemented as an actor creates or "spawns" other actors in response to receiving a message. After the creation of the user's service, the message is forwarded to this computation for processing.

These components include also cross-cutting concerns or aspects such as capability accounting, namely the monitoring of capability delivery and use. For example, both α *service CURLs* and α *user services* address capability accounting aspects, the former to keep track of issued CURLs, and the later to monitor capability use by users.

The described architecture is the *initial* architecture of COASTmed (at t_0). This architecture

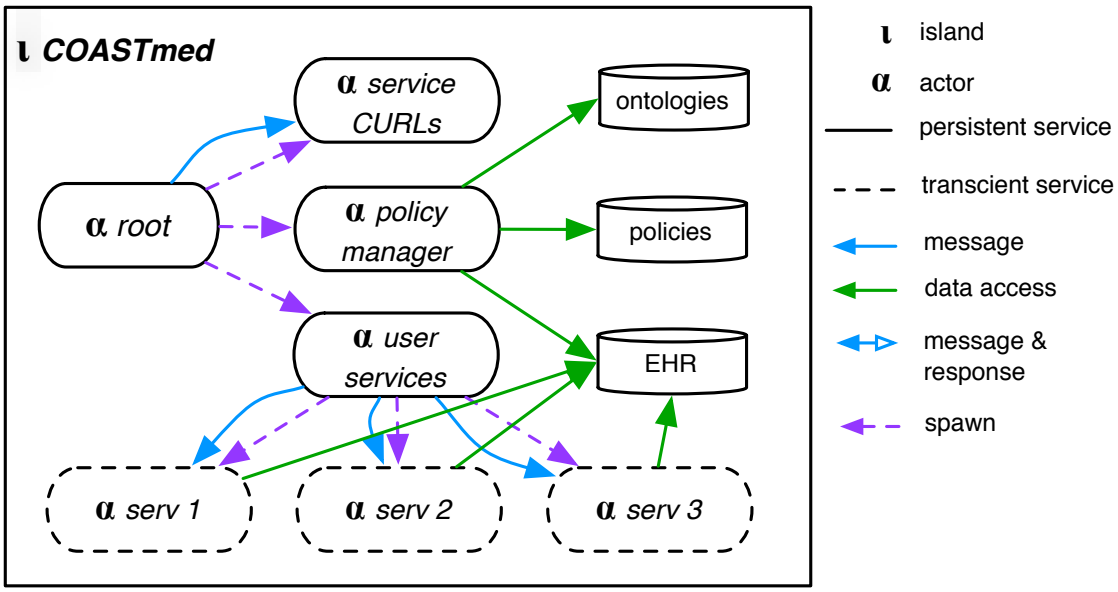

Figure 7.3: System architecture at t_n.

dynamically changes in response to both the organization's expanding needs and based on users' requests which result in the spawning user-specific computations (figure 7.3). In our current implementation of COASTmed, all user services exist within the same island. However, a provider may choose to allocate services within different islands according to some criteria. For example, a healthcare provider may choose to spawn critical services, such as those with more expansive access to health records, within more secure servers, or services for users who may run more computation intensive data analyses within more powerful servers.

7.3 Specifying Policies in COASTmed

In COASTmed, access control to services is performed based on policies set forth by a data provider such as a hospital, healthcare professional, or an overseeing health policy making agency. Privacy and operational policies in the context of EHR management and healthcare more generally may define the context and rights to access, create, and modify patients' data, supervise billing and insurance, manage medical supplies, coordinate the use of human and physical resources such as physicians, equipments, operating and recovery rooms, and so on (see related research on the healthcare resource management in [221] and [269]).

Policies are thus specified by an authorized policy maker such as a hospital administrator or an internal auditor. From our perspective, this principal is just another service user whose assigned capabilities include the ability to create, modify and delete policies. Therefore, at t_0 the single policy that enables creating all the system's policies, and consequently all users' services, is "a principal in the role of policy maker has the right to create policies." This right is associated to a binding environment who has two capabilities: `policy/new` and `predicate/new` which enable creating policies and policy predicates correspondingly.

In COASTmed, policies are represented in Rei, stored in database, and translated to the underlying implementation language for executing policy evaluation. To provide a concrete example of how a policy is created, consider the policy "a physician has the right to access a patient's electronic health record if and only if the physician is the patient's primary care physician." To specify this policy, the policy maker sends a custom closure to COASTmed

```
(lambda()
  (policy/new
    "A physician has the right to access a patient's electronic
   health record (read and modify his/her EHR)."
    "physician"
    "EHR/access"
    "patient"
    null
    (list (predicate/new "If the physician is the patient's
   primary care physician." "primaryCarePhysician(*physician*, *
   patient*)" #t))))
```

This lambda expression includes the `policy/new` function, which takes as arguments the policy's natural language description, the subject (`physician`), the authorized action (`EHR/access`), an optional policy direct target object (`patient`), an optional indirect target (`null`, no indirect target in this policy), and zero or more conditions. Conditions are created using a second function `predicate/new` which takes as arguments a predicate description, a predicate expressed in Rei (`primaryCarePhysician(*physician*, *patient*)`), and a boolean value depending on whether the predicate is positive or negative (`#t`). Although our approach does not allow prohibitive policies, we do consider negative policies to express exclusions and exceptions. For example, the description of the negative version of the example's predicate is "If the Physician is *not* the Patient's primary care physician". The consequence of this policy is granting access to all EHRs to physicians with exception to

those for whom they are the patient's primary care physician.

The α *policy manager* then stores the policy and predicates in the database. Policy rights and predicates are stored separately in different database table for predicate reuse, so that multiple policies can include the same condition. Along with the Rei predicate, α *policy manager* stores its translation to an expression in the underlying programming language—Racket in our case—for future evaluation so that compilation is done once for each predicate. For instance the predicate `primaryCarePhysician(*physician*, *patient*)` is translated to the lambda expression

```
(λ (args) (primaryCarePhysician (cadr (assoc 'physician args)) (
   cadr (assoc 'patient args))))
```

which replaces the variables `'physician` and `'patient` with constants provided through an association list `args` of key-value pairs, and then invokes the function `primaryCarePhysician` on those replaced variables.

Figure 7.4 describes the database structure for policies and predicates. Note that the field for policy conditions includes references to predicates in the predicates table.

POLICIES

ID	DESCRIPTION	SUBJECT	ACTION	TARGET	CONDITIONS
1	A physician has the right to access a patient's EHR	physician	EHR/access	patient	1

PREDICATES

ID	DESCRIPTION	REI	RACKET	POSITIVE?
1	If the physician is the patient's primary care physician	primaryCarePhysician(*physician*, *patient*)	(primaryCarePhysician physician patient)	#t

Figure 7.4: A relational database structure to store policies

This policy can be retrieved in Rei notation for policy analysis and comprehension

```
policy(physician,accessEHR,patient,null,(primaryCarePhysician(*
    physician*,*patient*)))
```

It is not expected that policy makers write programming language closures in order to create and modify policies. Custom user interfaces that include form-like fields correspondent to individual policy elements can be provided to non-technical policy makers. For instance, the COASTmed prototype provides a basic user interface to specify policies through a web browser (figure 7.5). A service client underlying the user interface dynamically composes these complex closures based on the policy maker's input. In addition, visualization tools can be provided to view and organize existing policies, as well as to filter them based on the action, subject, or direct or indirect target. For example, a policy maker may want to view all policies that bestow the right to order lab tests or those policies that apply only to nurses, and so on.

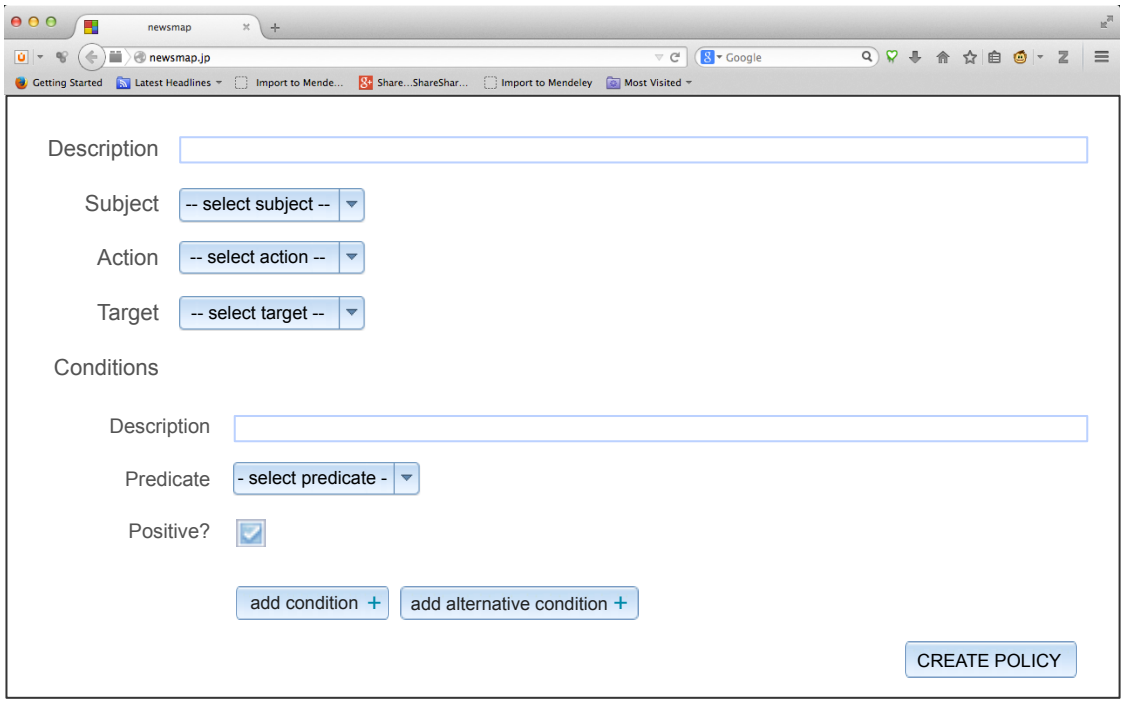

Figure 7.5: User interface to specify policies

Figure 7.6 graphically describes the policy creation process in COASTmed using a web-based user interface. This figure depicts two separate islands, where service provider ($I\,COASTmed$) and service user ($I\,policy\,maker$) are autonomous. COASTmed thus may have both internal and external users with respect to their affiliation with the provider. The policy maker, for example is an internal user of COASTmed, while a patient is an external user.

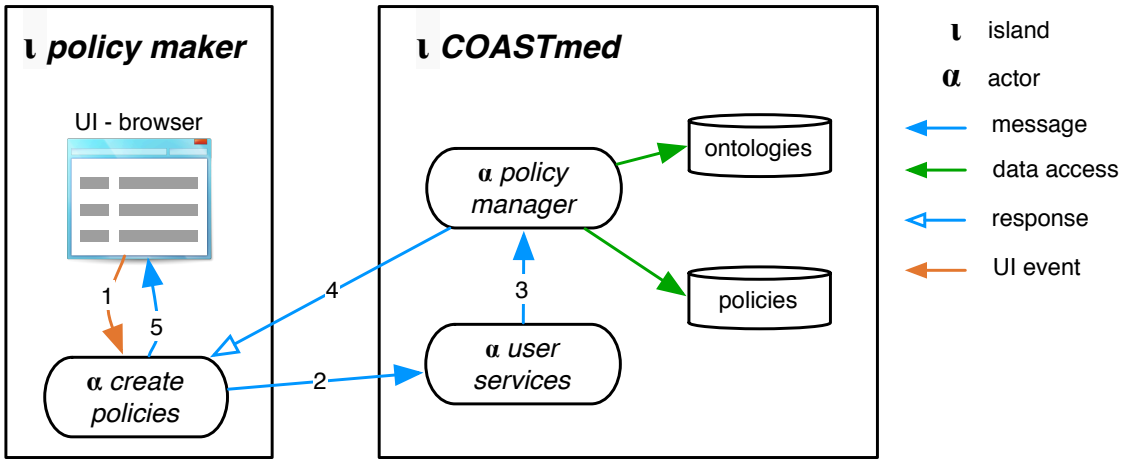

Figure 7.6: Policy creation process

7.3.1 Describing Policies Using Ontologies and User Interfaces

In 6.1.1 we outlined the relationship between ontologies and policies. Ontologies provide vocabularies to capture knowledge and concepts within a domain. In the healthcare domain, ontologies are particularly critical given the vast amount of existing domain terms. For instance, Puri et al. argue that no single ontology is sufficient to capture the growing needs of this domain and that integration and mapping between multiple ontologies is necessary [217]. To this end, COASTmed currently includes the LOINC (clinical observations) and the ICD (diseases classification) standard healthcare ontologies, and we seek to integrate other standard ontologies in future work.

Although our approach does not advocate for a specific way to represent ontologies, COAST-

med's system architecture includes a database management component for storage and retrieval of ontology instances (see figure 7.2). Different ways have been proposed to capture domain ontologies within a database. In COASTmed, a database table is an ontology class (e.g., table *employee*) and each column name is a class property (e.g., *name* and *age*), concept adopted from [25][176]. Class properties can be used to establish relations. For example, the *name* relation may associate a principal (identified through an unique identifier) and the name John. These relations establish domain facts, which are essential for policy evaluation.

In the context of our work, ontologies provide a vocabulary to specify policies, thus policies and conditions (predicates) can be composed using these standard concepts. Although our research focuses on the backend functionality of access control techniques rather than on the user interface aspect of COASTmed's clients, we provide some insights on how a policy specification tool interfaces with an ontology service for specifying policies appertaining to a domain. Figure 7.7 shows how the drop down boxes from 7.5 can be populated with domain ontologies. The second argument to the "diagnosis" predicate is selected from the ICD10 ontology maintained in COASTmed's database.

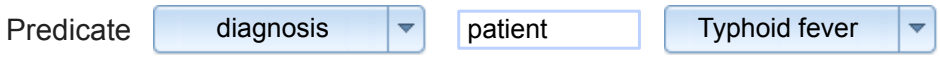

Figure 7.7: Using ontologies for policy specification

There are other lines of research from which to borrow ways to reason about, navigate through, and visualize ontologies within the user interface. For example, the WebODE ontology workbench [78] includes a tree-like ontology browsing, which may provide a selection alternative to drop down boxes for filling in policy forms such as in figure 7.5.

7.4 Generating Service CURLs

In order to use COASTmed's capabilities, a user or consumer needs to first obtain a user-specific service CURL as described in 6.1.4. Figure 7.8 illustrates the sequence of steps for a CURL to be issued to a user.

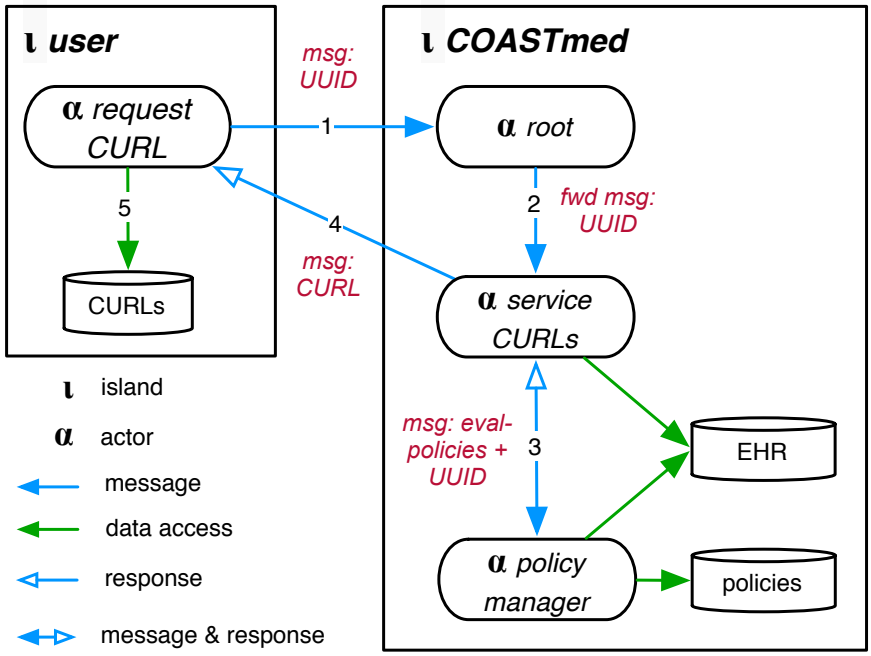

Figure 7.8: A service consumer's service CURL request

First, the service consumer sends a message

```
(curl/send @COASTmed (service-request/new 12345 @me))
```

where: (a) `curl/send` is COAST's primitive to send a message to another computation; (b) `@COASTmed` is the public curl addressing the COASTmed service provider; (c) `service-request/new` is a message type understood by the service provider, requesting a new service CURL; (d) 12345 is the user's UUID which may be a social security number, a public key, or any other unique identifier; (e) `@me` is the reply-to CURL to which the response (which may contain a service CURL) needs to be sent.

The root computation α *root* at COASTmed receives this message, and interpreting the message type forwards the request to α *service CURLs*, the specialized computation whose responsibility is to create service CURLs. Next, α *service CURLs* looks up the EHR database to retrieve the set of roles (if any) that user 12345 fulfills in the system. For example, this UUID may belong to Dr. Smith, who has the *physician* role.

The user's identification number along with the roles fulfilled in the system are sent to the α *policy manager* computation, which retrieves all the policies relevant to the user's roles. In the previous example, the retrieved policies will be those whose subject is either *physician* or any if its parent roles. A parent role of *physician* is, for example, *employee*. Thus a policy that bestows a right to a hospital employee gives the same right to all physicians, since a physician is a specific type of healthcare employee. For example, consider the policies in table 7.1 (taken mostly from appendix A). The relevant policies for Dr. Smith are P2, P3, P4, P5, and P11. Note that P11 is selected as a relevant policy since *employee* is a parent role to *physician*.

7.4.1 Evaluating Policies at CURL Creation Time

The relevant policies retrieved are then evaluated by α *policy manager*'s evaluation function, which takes as input a list of policies and an association list of arguments (i.e. `(list (list k1 v1) (list k2 v2) ...)`). At this stage, the only known variable value is the principal's identity, therefore the conditions' variables which match the policy subject are associated to the user's UUID. In this example, the arguments input to the evaluation function are `(list (list 'physician 12345) (list 'employee 12345))` since the relevant policies' subjects—physician and employee—are associated to Dr. Smith's UUID.

Those policies that do not include conditions (P4, P5, and P11) automatically evaluate to *true*. Alternatively, for those policies which include conditions (P2 and P3) the list of condi-

	description	subject	action	target	conditions
P1	Only clinical psychologists can update patients' psychological evaluation.	clinical psychologist	PhsycEval/update	patient	
P2	Medical specialist doctor may be allowed to access the results of sensitive medical test.	physician	SensTestResults/read	patient	specialist(*physician*)
P3	A doctor can access a patients health record when the patient has designated him as his/her primary care physician.	physician	EHR/access	patient	primaryCarePhysician(*physician*, *patient*)
P4	A specialist physician (e.g., cardiologist) may prescribe drugs if the treated illnesses is related to his/her specialty (e.g., cardiac- related).	physician	Drug/prescribe	patient	
P5	Physicians may order laboratory tests.	physician	Tests/order	patient	
P6	Hospital pharmacists have only read access to prescription files.	hospital pharmacist	Prescriptions/read		
P7	Nurses can make a housekeeping request.	nurse	Housekeeping/request		
P8	The patient has the right to inspect and obtain a copy of the medical records.	patient	EHR/read	patient	
P9	A patient's insurance company may access medical procedure information for billing purposes.	insurance company	EHRprocedures/read	patient	insuranceCompany(*insurance company*, *patient*)
P10	Neonatal nurses can update infants health records.	nurse	EHR/update	patient	specialization(*nurse*, neonatal) and newborn(*patient*)
P11	Employees can enroll in provided skill development programs.	employee	DevProg/enroll		

Table 7.1: Sample policies

tion variables is retrieved; recall that in Rei's conditions, variables are delimited by asterisks (*). For each policy predicate, the list of variables is checked against the argument list for a match in order to replace variables with argument values. If any of a policy's conditions has unresolved variables, for example "patient" in P3's condition, the policy evaluates to *'not-evaluated*. The reason for unresolved variables is because there are some contextual variables that depend on the specific service request. For example, consider the condition `later(*current-time*, "12:00 P.M.")`, which constrains access to some function depending on the time of day it is invoked; the value of `*current-time*` is unknown until the action is invoked.

Instead, if all the policy conditions' variables have a corresponding value according to the argument list—namely if there exists a value for every predicate variable—the evaluation function is executed for each policy condition. This function takes two values as input: the argument list (the association list of variable names and known values) and the database-retrieved implementation language (i.e. Racket) version of a Rei policy condition. Variables in the Racket predicate are replaced with values and each predicate is executed just as any programming language function to obtain a truth value. For example, the Racket version of P3's condition `specialist(*physician*)` is `(lambda(args)(specialist (cadr (assoc "physician" args)))`. When this predicate executes, the provided argument list `args` is used to match the variable `physician` with its associated value. Therefore, in the example, what is ultimately executed is `(specialist 12345)`. This predicate returns `#t` or `#f` depending on whether user 12345 is a specialist physician or not.

For policy execution, it is necessary that all its condition predicates exist within α *policy manager* execution's environment as functions which return a boolean value. Therefore, α *policy manager* has to be able to reach a function

```
(define specialist
  (lambda (uuid)
```

```
      ......))
```

which accesses the EHR database to check a fact and return a truth value.

Given that Dr. Smith is not a a specialist physician but a general practitioner, the function `(specialist 12345)` evaluates to `#f`. Consequently, P2 evaluates to false. Recall that if there are more than one condition in a policy, it only takes one evaluating to *false* for the policy to be *false*. Also recall that for composite alternative conditions—e.g., if subject is either a nurse or a physician—only one needs to return *true* for the policy to be *true* (assuming other simple conditions are *true* as well).

To recap, the result of the evaluation of the policies in 7.1 for user 12345 is as in table 7.2.

policy	eval result
P2	F
P3	'not-evaluated
P4	T
P5	T
P11	T

Table 7.2: Evaluation of sample policies

The evaluation results are returned to α *service CURLs*. Policies that evaluate to `#f` are discarded (i.e. P2) and the remaining policies are matched with their associated capabilities as shown in table 7.3. As described in section 6.1.3, first, implicit capability associations are made by searching in the global binding environment for a binding which has the same name as the policy action. If a match is not found for a particular policy action, the database is consulted for explicit associations (since policy action and corresponding function may not have the same names). In the current implementation of COASTmed, capabilities are reified as binding environments that may contain one or more functions.

policy	capability
P3	EHR/access
P4	Drug/prescribe
P5	Tests/order
P11	DevProg/enroll

Table 7.3: Sample policies and capabilities associations

Following, a CURL addressing *α user services* is created for user 12345 (Dr. Smith), which contains as metadata the set of authorized capabilities as well as the unevaluated policies. Metadata is implemented as a hash table, where keys are `'UUID`, `'environments` and `'policies` and their corresponding values, for this example, are `12345`, `(list SensTestResults/read Drug/prescribe Tests/order DevProg/enroll)`, and `(list (vector P3 physician EHR/access patient (list (primaryCarePhysician physician patient))))` correspondingly. This CURL is graphically described in figure 7.9.

issuing island	path	expiration date	use count	uuid	authorized capabilities	policies	digital signature
www.EHR.com 127.0.0.1 5001	null	10e100	10e100	12345	EHR/access Drug/prescribe Tests/order DevProg/enroll	P3	IQB1AwUBMVSiA5J...

Figure 7.9: CURL sent to Dr. Smith

Finally, this CURL is issued to the user by sending a message to the computation addressed by `@me` in the user's request; the user stores this CURL for future use. If instead, all policies evaluate to `#f` , no user CURL is generated since it means that the user currently has no rights with respect to COASTmed's services, therefore an error message is sent to the user.

This CURL is conceptually similar to the token proposed by Gavriloaie et al. [118], namely a "non-transferable token that she can use to access the service repeatedly without having to negotiate trust again until the token expires." Access control by way of CURLs is different from traditional access control in many aspects. Although policies could be repeatedly checked with traditional role-based access control, narrowing down the space of policies that

have to be repeatedly checked (i.e. those embedded within CURLs) in domains where there are potentially hundreds of roles, policies, and users can improve performance and reduce the overhead. In addition, traditional access control is more focused on the graphic user interface by allowing buttons and menus corresponding to specific actions to be present or absent according to the user's authority, but here we are dealing with access control with regards to mobile code execution which is not meant to be used by end users but by application programmers. Therefore, access control through CURLs works at a lower level than traditional access control. Also, having metadata regarding the authorized capabilities also informs consumers what are the base building blocks which can be used to build more complex custom services. Moreover, CURL-specific code can be embedded to interpret the user's message. Lastly, the way traditional access control works is through a number of *if else* statements scattered around the code to "show/hide" different secured elements. This is error prone and is certainly less secure than providing users with personalized and bounded services, where unauthorized capabilities are simply absent and unreachable.

7.4.2 Capability Accounting

In section 6.1.7 we argued that service providers may want to keep track track of the CURL issued to different users in order to augment, constrain, or revoke previously bestowed rights. If at least one policy is evaluated to either *true* or to *'not-evaluated*, in other words if a user CURL is generated, α *service CURLs* records information about the issued CURL in a database. The information recorded is the address the CURL was issued to, the issued CURL's unique identifier, the bestowed capabilities, and whether this CURL has been revoked. The "revoked" field will be accessed at the user's requests to determine whether the CURL has been or not revoked, and accordingly process or reject the user's message.

7.5 Using Services through User-specific CURLs

Having been issued a CURL, a user can make use of it to access the capabilities authorized to him/her. Figure 7.10 shows the sequence of steps for consumers to use COASTmed services. First, the user application retrieves the CURL `@COASTmed` from its private CURL repository in order to send a message to the service. A message may contain primitive values, binding environments, and closures that are interpreted by the provider. Leveraging the capabilities bestowed, information included in the CURLs metadata (see figure 7.9), Dr. Smith sends a closure `(lamda()(EHR/access 54321))` through a user interface to COASTmed to obtain a patient's health record. When α *user services*, the computation which `@COASTmed` names, receives a user's message it first checks for three conditions: that the used CURL has not been revoked, that the CURL's use count has not been exhausted, and that the CURL is not expired. The second and third conditions are automatically handled by the COAST infrastructure. If these three conditions are met, α *user services* retrieves the list of unevaluated policies from the CURL's metadata. If the list is empty, then α *user services* creates a new user service to serve the user's request. Otherwise, a second stage of policy evaluation for the unevaluated policies takes place.

7.5.1 Evaluating Policies at Service Use Time

Recall that user-specific CURLs may include, as metadata, policies that remained unevaluated at CURL creation time (see section 7.4.1). The reason for having this second round of evaluations is that the authority to use some capabilities may depend on the context of the specific service use instance. These are for instance environment factors such as the time and date or the specific action's targets (i.e dependent on a function's arguments). For example, the target to an action `PATIENT/prescribe` is *patient*; a condition to prescribe medicines to a patient is that the physician who prescribes ought to be the patient's pri-

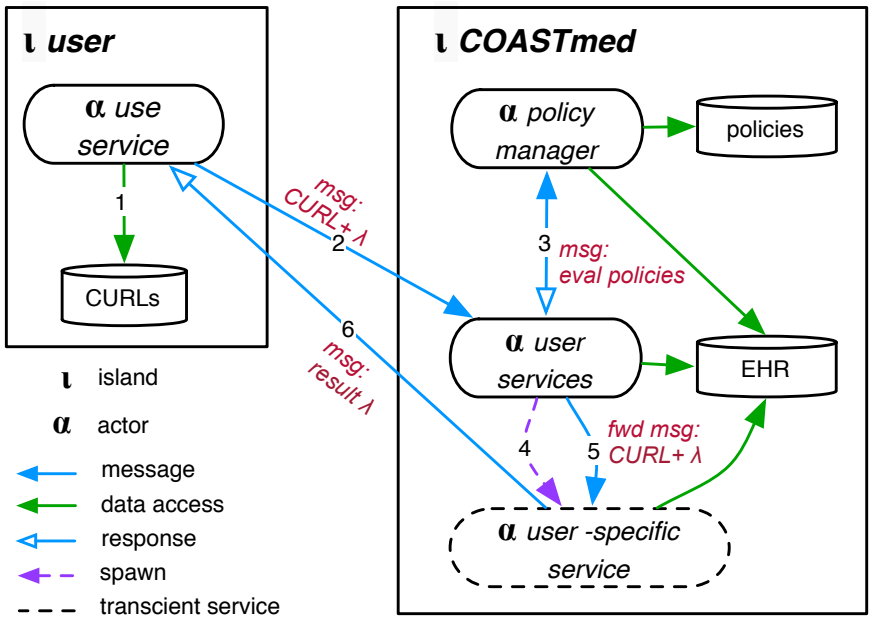

Figure 7.10: Dynamic user-specific service creation

mary care physician. Another example is where the patient needs to explicitly consent to a specific physician treating him; information regarding consent is retrieved for the specific patient that is subject to the user's action from the fact base. COASTmed currently does not account for patient-specific policies with respect to their data, but it requires little effort to also including patient specific policies that are combined at run time with the provider's own policies to provide a more patient-controlled access to his/her data.

At an incoming user message, α *user services* sends α *policy manager* those unresolved policies for re-evaluation only if they are relevant to the user's message. To determine whether the policies within the CURL are relevant, the user's closure is inspected for matches between the used capabilities and the policy rights. If there are no matches, namely no capability that is relevant to any of the policies embedded in the CURL is being used in the custom closure, no policy evaluation is done and α *user services* creates a user-specific service to execute the closure.

Otherwise, if policy rights are relevant to the capabilities used within the closure, these

are evaluated in the same way as at CURL creation time (i.e. at the first stage of policy evaluation). Policy conditions are individually evaluated. The list of arguments that is provided to this process is as follows:

- the subject variable within each policy is matched with the user's UUID, embedded in the CURL's metadata (see figure 7.9);
- the policy's direct target is determined by the first argument of the corresponding function call within the user's message;
- the policy's indirect target is determined by the second argument of the corresponding function call;
- the remaining variables are evaluated within Racket's read-eval-print-loop (REPL) through reflection techniques; for example, the policy variable *current-time* is matched to an existing language variable `current-time` existing within α *user services*'s lexical scope.

For example, if a policy's subject is "physician", the action is "prescribe", the direct target is "aspirin", the indirect target is "patient", and a user's closure is `(lambda()(prescribe "aspirin" 4444))`, then the variable "physician" is matched to the UUID embedded in the CURL and the variable "patient" is matched to 4444 and replaced in all the policies' conditions.

Going back to the example, assume the user's closure sent to COASTmed is `(lambda()(EHR/access 54321))`. The right or capability which the unevaluated policy in the CURL (P3) provides authority to is "EHR/access". Since this capability is used in the user's closure then P3 is considered relevant to the user's request. Then P3's conditions are evaluated—in this case `primaryCarePhysician(*physician *patient*)`. The argument to the evaluation function is `(list (list "physician" 12345)(list "patient" 54321))`, where the

target "patient" is matched with the EHR/access function's first argument (i.e. `54321`). The Racket version of the condition is retrieved and it is executed by applying the associative list as an argument. The executed function is then `(primaryCarePhysician 12345 54321)`.

This policy condition evaluates to *true* if, according to the EHR database, Dr. Smith is the primary care physician of the patient with UUID 54321, and *false* otherwise. If it evaluates to *true* the the user's message (with the embedded custom closure) is sent to a newly created user service for execution. Otherwise, if P3 is *false*, meaning that Dr. Smith is not authorized to access this patient's record, the user's message is not processed and an error message is in turn sent to the user.

The EHR database plays a primary role in the evaluation of policies, since it is where all the organization's facts live. Predicates corresponding to policy conditions, when executed, retrieve information from the database to obtain a truth value. Organizations can choose how to organize and store this information (e.g., alternatively in XML files), and predicates are proxies between policies and these facts.

For further clarification on the two stages of policy evaluation, we describe in table 7.4 the differences between the first and second stage policy evaluation processes.

7.5.2 Other Explored Techniques to Retrieve Facts

Recall that to evaluate policies it is necessary to establish and retrieve facts about the relationship between concepts. We explored other alternatives before choosing to implement conditions as predicates that retrieve information from a database. One of these experiments involved expressing explicit logic-based relations in Racklog, the Racket implementation of Prolog. In Racklog, we can define a relation *age* and declare relation instances:

	first stage	second stage
time of evaluation	at CURL creation time	at service creation time
evaluated policies	all those relevant to the user's roles	all those relevant to a user's message
source of the policies	COASTmed's policy database	user's CURL metadata
consequence of policies evaluating to true	capabilities authorized to user	if all evaluate to true, user's message is processed
consequence of policies evaluating to false	capabilities excluded from user's binding environment	user's message is rejected
consequence of unevaluated policies	policies stored in CURL's metadata for later evaluation	user's message is rejected

Table 7.4: Differences between first and second stage of evaluation

```
(define %age
  (%rel ()
    [('John 32)]
    [('Mary 40)]
    [('Lisa 21)]))
```

Another example is the relation "patientIllness", defined as:

```
(define %patientIllness
  (%rel ()
    [("John Doe" "chicken pox")]
    [("Jane Doe" "diabetes")]))
```

With these relations we can perform queries such as `(%which ()(%age 'Mary '60))` which returns `#f` and `(%which (what) (%patientIllness "John Doe" what))` which returns `'((what . chicken pox))`.

However, we found that this approach is not scalable since these relations are hard-coded and cannot be dynamically retrieved from a database.

In another experiment, we defined relations based on database structures. For example, the relation `patient_demographics.name` associates the primary key column of table "patient_demographics" to the table column "name". Therefore, the evaluation of the policy condition `patient_demographics.name(12345, Benites)` is either true or false depending on whether the last name of patient with id 12345 is Benites according to the database. At the implementation level we name this relations according to the pattern *[table name].[column name]*. However, this approach is too tightly coupled to low-level backend components, making it difficult to scale and adapt to other sources of information, and exposing the database structure may pose security issues.

Other researchers have also worked on similar policy translations. For example, Samuel et al. transform OWL and Semantic Web Rules (specified in SWRL or RuleML) into Prolog for automated reasoning on ontologies, knowledge bases, and rules [230]. There is also an ample research field and technologies for representing knowledge. For example, the LOOM knowledge representation system which builds concept taxonomies over which inferences can be made [177]. A LOOM concept is, for example, $\texttt{Man} \equiv \lambda\ \texttt{x.Male(x)} \wedge \texttt{Adult(x)}$, where a subject needs to fulfill both predicates (`Male(x)` and `Adult(x)`) to be classified as a man.

7.5.3 Dynamically Creating User Services

As previously described, there are three situations in which α *user services*, conditional on the user having a valid CURL, creates a new user-specific service:

- if the policy list in the CURL's metadata is empty;
- if there are not any relevant un-evaluated policies (embedded in the CURL) to the user's message (none of the policies' capabilities exists in the user-formed closure);
- there are relevant policies to the user's message, but they all evaluate to *true*.

To dynamically create a user-specific service, α *user services* retrieves the authorized capabilities information from the CURL's metadata and creates a new binding environment which has as key-value pairs the functions' names and their corresponding lambda expressions. This environment may also contain primitive values and data structures. Keys are retrieved from the CURL's metadata regarding the authorized capabilities, while their values, e.g., the corresponding functions' syntax, are provided by the underlying language's evaluation handler.

α *user services* jumpstarts in a separate thread of computation a closure whose lexical scope is defined by this binding environment. The closure usually involves a loop which receives and executes messages, and performs capability accounting as described in 7.5.5. Figure 7.11 shows the relevant aspects of dynamic service creation based on CURL-embedded data.

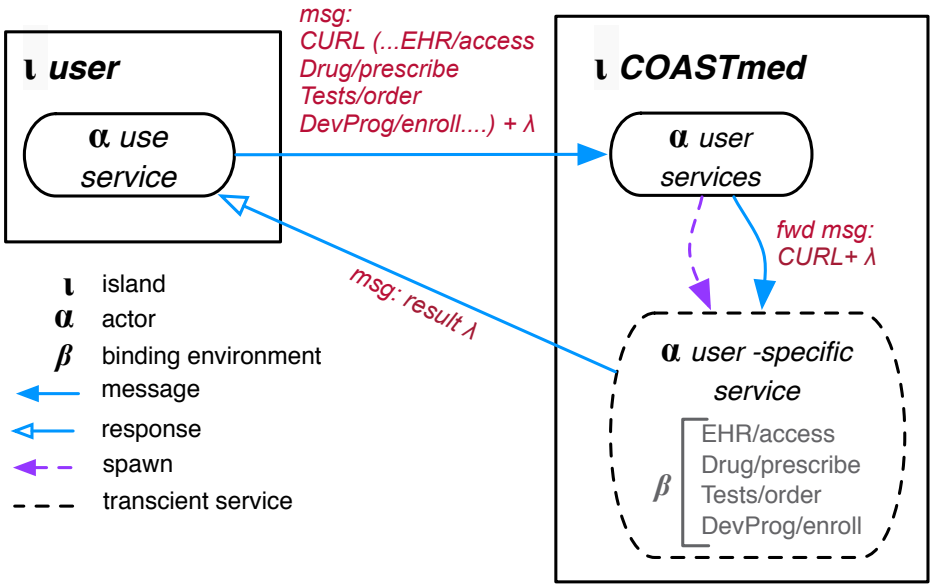

Figure 7.11: Dynamic service creation based on CURL's data

7.5.4 Typed Messages

As prescribed by COAST's principles, the interpretation of a message delivered to computation y via a CURL u_y is u_y dependent. However, to increase shared understanding between service consumer and service user, COASTmed supports message types. For example, to request a user CURL, a user uses message type `service-request/new` and for using the service through a CURL the message type `service-call/new` is used. Other message types examples are `CURL` to send a CURL to another computation or `error/new` to send and error message through type. Message typing is also used in the COASTcast prototype [121], the first COAST application.

Message types generate structured tuples that allow COASTmed to interpret users' intent. Any island or computation can define their own message types; whomever wishes to communicate with this computation needs to use the available computation-specific message types. A message that leverages a message type is, for example,

```
(curl/send @COASTmed (service-request/new "123" @me))
```

which sends a message to COASTmed using CURL @COASTmed to request a service CURL for user with UUID 123 by way of message type *service-request/new*. This way, a computation in the COASTmed can route this request to the computation responsible of generating and issuing CURLs (e.g., α *service* $CURLs$ in figure 7.8).

Another way to support message types and achieve the same effect is by leveraging a CURL's *path* which is basically a list of symbols. These arbitrary, provider-defined values can be used to interpret incoming messages. An example is provided in [123]:

```
(let* (.....
  (path (list 'books 'sale))
  ....)
(curl/new sale path metadata))
```

However, there is subtle distinction between message interpretation through CURLs' path values and message type tuples. Since a path's values are fixed, the interpretation of a message based on those values is always the same since the data in the CURL cannot be changed or tampered with. In contrast, message types provide a more flexible technique for message interpretation—the provider declares a set of message types that it can interpret and accordingly the user uses these message types to communicate with the provider. The choice of message type depends on the user's needs with respect to the system. These different message types can be included in a CURL's metadata or can be advertised out-of-band. However, both CURL paths and message types can be orthogonally used according to the provider's own rules. For example, a CURL's path `(list "student" "ics")` may identify a user as an ICS student, while message types `class/enroll` and `class/drop` allow enrolling and dropping classes correspondingly.

The benefit of using typed messages is to allow the provider to understand the user's intent. Recall that the service provider defines what are the permitted message types and that, as dictated by the COAST style principles, the interpretation of a message depends on the receiving computation and on the CURL used to send the message, so no security property is violated though the use of message types. If a user maliciously or unintentionally misuses a message type (e.g., sends a closure along in a `service-request/new` message type, when it should send a user's UUID), the message will simply be rejected since the message can not be correctly interpreted.

7.5.5 Capability Accounting for Tracking Service Usage

Capability accounting is also performed when the user successfully or unsuccessfully tries to use his/her user-specific service. The rationale is to allow the service provider to keep track of how the bestowed capability is being used and whether the user's message was rejected

and why. There are three stages at which capability accounting can take place after a user has already been issued a CURL:

1. when α *user services* receives a user message that remains unexecuted due to policy non-compliance (i.e. not all the policies relevant to the message evaluate to *true*);
2. when one or more bindings within the user's closure are recognized or exist within the service's scope. This may be possibly due to a syntactic error in the custom closure or due to a malicious attempt to use an unauthorized capability;
3. when the user's mobile code executes successfully.

Therefore, capability accounting includes both successful and unsuccessful interactions with the service. The information stored in the database for capability accounting purposes is the CURL's resource key (a unique identifier), the closure sent using the CURL, the date, a boolean value stating whether the closure was executed or not, the reason, and the failed policy if such reason was because a policy evaluating to false.

Since a principal using a given CURL to access a service may not be the user to whom the CURL was issued (the CURL may have been shared with a third party or the CURL may have been obtained in an unauthorized or illegal manner), a provider may want to also record, for capability accounting purposes, the island address of the consumer computation. Having this information may enable the provider to curtail access to the service if the message sender is unknown or if it has been recognized as an untrusted party. Although the current version of the COAST infrastructure does not support obtaining the service user's address, the forthcoming version of the infrastructure will allow the collection of this information based on the TCP connection information.

7.6 COASTmed within a Large Organization

COASTmed—a service provider—constitutes an individual, decentralized island which provides a set of capabilities and information to a group of users. However, COASTmed may be one of many systems/islands within a larger organization. Therefore, COASTmed is part of a system of systems.

Take for example a large organization such as Kaiser Permanente which is a consortium of many regional hospitals and medical groups; some of them belong to Kaiser while others are owned by individual physicians in partnership with Kaiser. In this context, COASTmed is an individual system managed by one of Kaiser's subsidiaries which offers a set of capabilities within the organization. Each subsidiary then manages, so to say, their own COASTmed. Individual health record systems have no knowledge of other systems and databases, but exist and behave autonomously.

Now, Kaiser may decide that all of its subsidiaries have to comply with a policy that states that clinical researchers must be approved for access to medical information not only at a local level, but by an organization-wide auditor. Therefore, each local system needs to somehow be aware of this policy so that all Kaiser's affiliated services are governed by the same policy. In such a case, an organization-level policy maker can define this global policies.

There are two ways to go about implementing these global policies within local systems. The fist approach is to provide a "master" policy maker with the right to create new policies in each local system. This means that this individual needs to be registered as an authorized system user in every system, hold the role of *policy maker*, and the policy

```
policy(policy maker,POLICY/create,null,null,null)
```

needs to exist in every local policy database. Therefore, the master policy maker has a CURL for each subsystem that allows her to create policies. The policy maker through a

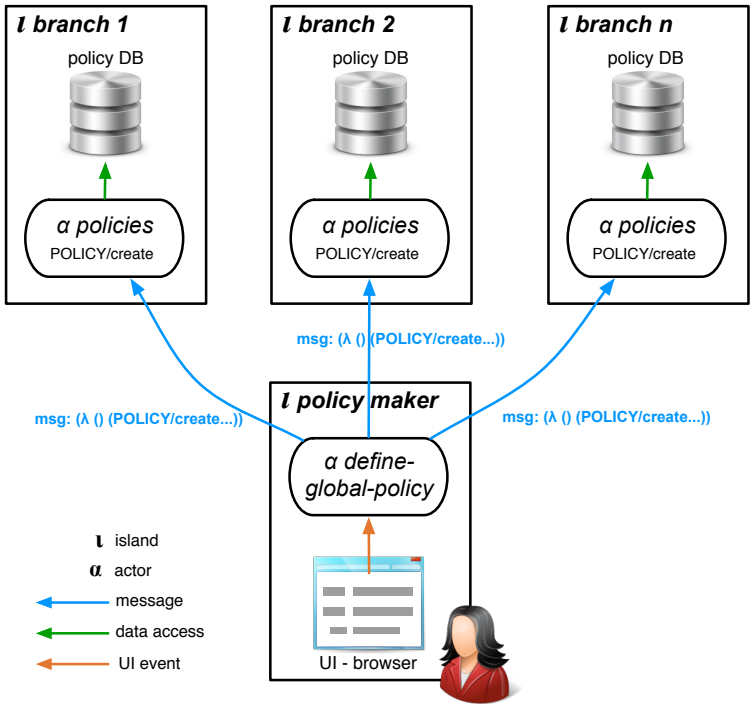

Figure 7.12: A master policy maker creating a new global policy within each service provider system.

user interface (such as the one presented in figure 7.5) can define policies. An underlying COAST computation then sends the same policy-creation message to all Kaisers's branches as described in figure 7.12.

A second alternative is to have all individual providers retrieve these organization-wide policies from a central database. In this scenario, the CURL creation process described in section 7.4 would not only retrieve policies for a particular user role from the local database, but also from the global policy database (figure 7.13).

From a manager's perspective, both approaches achieve the same goal. However, these implementations have technical tradeoffs that may have consequences in the ability to enforce these policies. For example, the second approach has less overhead in terms of creating global policies since these are stored in a single, organization-wide database. The drawback is however a central point of failure—if the central policy system is down for some reason no provider can implement such global policies. The first approach instead has more com-

putational overhead since the same policy needs to be stored in multiple policy databases, however the enforcement of such policy does not depend on the reliability of a central policy providing island. In addition, this approach respects decentralization, which is at the core of this domain scenario. Also, the global policy maker can verify that local branches are enforcing this policy by periodically checking whether the defined global policies are effectively within the policy database of each local system by sending each local provider a message `(lambda()(policy-exists? policy))` and expecting all of them to respond `true`.

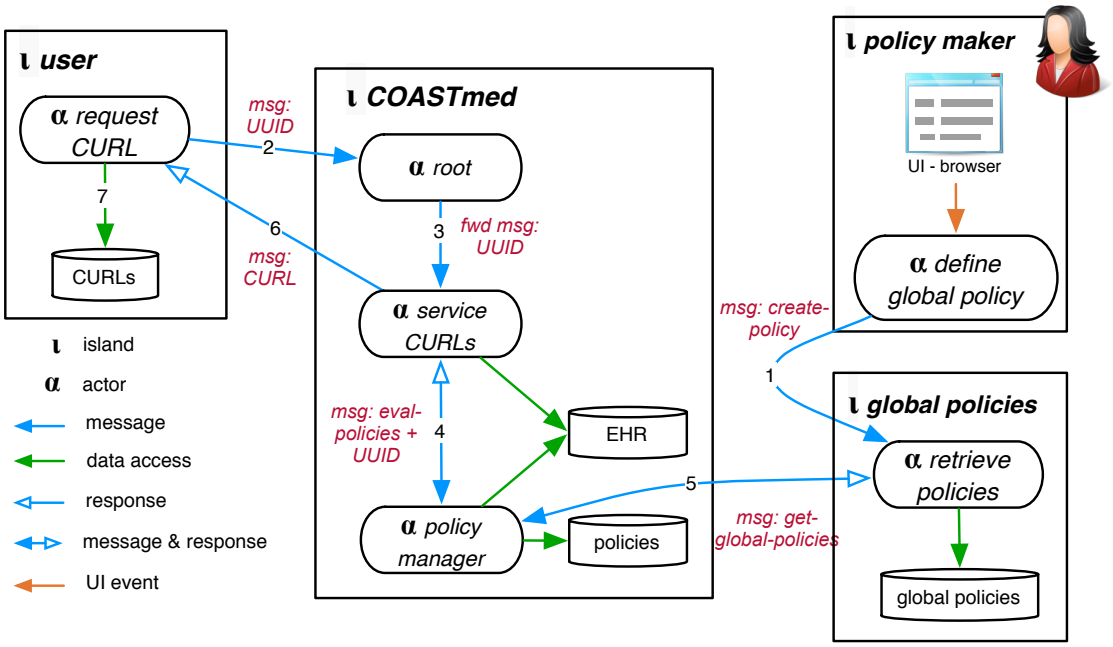

Figure 7.13: Policies also being retrieved from a global database at CURL creation time.

7.7 Implementation Technologies

COASTmed leverages Motile/Island, the existing COAST infrastructure. Although architectural styles are detached from implementation details, this infrastructure is provided to assist in the development of COAST applications. This implementation enforces developers to comply with the communication and computation execution constraints imposed by the COAST architectural style. Motile is a language for the serialization and exchange of mobile

code among COAST peers. Islands are the implementation of peers—single address spaces identified by an DNS name, a port number, and a public key—which host one or more actors, the reification of COAST computations. Islands provide the required management and communication infrastructure—including CURLs—for computations to execute and communicate. Motile allows defining both computations and the closures exchanged. The main components and connectors in Motile are thus actors, binding environments, CURLs, islands, the compiler, and the serializer. Motile is written in Racket, an implementation of the Scheme functional programming language. An in depth conceptual and technical description of Motile/Island can be found in [122].

To specify policies, we leverage the Rei policy language, whose foundations on deontic logic allows describing policies in terms or rights, prohibitions, obligations, and dispensations. Policies are rules associated to subjects. Although Rei is domain independent, policy specification relies on domain vocabularies. Rei was chosen among other policy languages after thorough evaluations (section 5.2) due to its compact, well-defined, and expressive logic-based syntax for describing healthcare policies, as well as its natural compatibility with COAST. Kagal et al. developed a policy engine implemented in the Prolog language to reason about policies specified in Rei. We have, however, chosen to translate Rei policies to Racket (COASTmed's underlying implementation language) due to implementation convenience so that conditions (implemented as predicates) can be executed at runtime. We explored a similar approach to Kagal et al. by expressing policies and relations in Racklog, which embeds Prolog-style programming in Racket [8]. However, and as we discussed in 7.5.2, this approach did not scale since predicates are hard-coded.

We also use HTML and JSON to provide some basic graphic user interfaces to interact with COASTmed computations. JSON enables exchanging data structures from COAST computations to the browser and vice versa. Lastly, we use the MySQL database management system to store and retrieve policies and domain data.

7.8 Design and Development Experience Insights

Our experience with designing and developing COASTmed provides valuable insights and recommendations for future design and development of COAST- and policy-based applications. It is first and foremost necessary to define what are the persistent internal and external services, the concerns they address, and their responsibilities within the system. A system's descriptive architecture involves conceptualizing these services and their interactions. Along these lines it is important to define the types of messages these computations will exchange. In COASTmed we leveraged a custom message type system that encapsulated messages in well defined, typed tuples that computations where built to process.

In addition, it is necessary to set forth the capabilities (including data availability) that will be exposed to users, as well as the user/role specific policies that enable the exposure of such capabilities as services.

One of the difficulties encountered in the implementation was leveraging the low level primitives and functions of the COAST infrastructure. To solve this challenge, which improved the usability of Motile/Island, we built a set of utility functions on top of these primitives to facilitate the creation and execution of computations, and the creation of Motile closures and CURLs. These utility functions deal with boilerplate code and enabled focus on the distinctive properties of individual computations. Having said that, an implementation framework such as Motile/Island is essential to implement systems that adhere to the COAST style. Motile/Island helps maintaining architectural consistency, namely that computation behavior and interaction abides by the principles set forth by COAST. Motile/Island takes care of these issues, allowing designers and developers to focus on the domain specific aspects that aim to solve specific problems.

Also, communication between COAST computations and user interfaces needs to be more closely addressed since the current implementation component dealing with graphic user

interfaces is fairly complex. There needs to be a better understanding about the different communication patterns between computations and front-end user interfaces. In addition, appropriate GUIs are extremely important for the effective specification of policies. Although our policy language based on Rei is more amenable to human understanding, therefore easier to manually specify policies compared to XML-based languages such as XACML, having an easy-to-use graphic interface that offers choice alternatives is more usable and prevents specification errors.

Developing COASTmed involved a steep learning curve; without simple examples of how to set up computations and how to send messages among each other it proved difficult to map architectural principles to implementation techniques. However, once we got the first computations up and running it was easier to use the COAST infrastructure, replicate this process, and scale the application. With COASTmed as a reference implementation we aspire to provide future design and development guidance for other COAST-based applications and more insight on the formal specification of access control policies.

The key benefits of using COAST as the architectural style for the development of COASTmed and in general for service oriented applications are:

1. flexibility and dynamicity to create services on the fly by combining at will an organization's available assets;

2. security through the concept of capability with respect to a service as well as the "boundedness" of services, where unauthorized functions are not reachable within the service's lexical scope;

3. expressiveness to create more complex, user-defined services through functional composition and mobile code;

4. focus on decentralization to allow multiple autonomous parties to be both service

consumers and providers, and securely collaborate in complex data sharing processes.

Therefore, COAST was essential for the development of COASTmed and more generally to obtain properties of user-differentiability and customizability. Perhaps these properties could be obtained with other technologies, however this would involve more complex solutions based on a set of multiple architectural styles (e.g., mobile code, event-based) and technologies (e.g., message buses, code interpreters, sandboxing, mobile agents), while COAST represents a "one stop" solution to these challenges.

Furthermore, the benefit of combining COAST principles and formal policies is the ability to (a) dynamically create user-specific services according to upfront and well defined criterion (captured in policies), and; (b) adherence of services' behavior to access control policies.

Chapter 8

Evaluation

The main, high level question that we seek to answer is "does our approach enable the different participating parties in a given domain to access the information and computation capabilities they need, in the way and at the time required, and within the boundaries imposed by the service provider by law or for the sake of privacy?" The aforementioned boundaries limit users' authority by, in turn, disallowing the access to information and capabilities they are not entitled to use.

Our hypothesis is that the enabling technologies to achieve this secure yet flexible access to data and computation are services whose essential properties are the provider-controlled differential service provision and the user-controlled customization which are simultaneously enabled through solid architectural principles based on capability-based computational exchange in combination with formal policies.

We evaluate our conceptual and technical approach towards testing this hypothesis (described in sections 6 and 7) through practical experiments and comparative analyses in the healthcare context. The choice of empirical domain is deliberate; healthcare involves a network of users and uses of patient data for various (often conflicting) purposes, uncountable

trust relationships, and complex legal and ethical strictures.

Our evaluation methodology comprises scenario-based evaluations (section 8.1) and qualitative comparative analyses (section 8.2). Scenario-based evaluations consider the context-dependent ability of a system to meet the desired properties [156]. For this stage, we COASTmed use as a testbed for scenario simulations.

Qualitative comparative analyses instead are carried out with systems and technologies approaching similar challenges in the healthcare and other domains with respect to the desired properties—differential access and customization.

8.1 Scenario-based Evaluations

Our research instinctively calls out for scenario-based evaluations which consider the context-dependent ability of a system to meet the desired properties [156].

We describe a set of bounded scenarios or "vignettes" which capture inter-agency processes in healthcare, our sample domain, involving information exchange and access control semantics based on complex data disclosure policies. Scenarios for these evaluations are collected from the literature on healthcare and EHR and more generally in the Software Architecture and Information Systems domains, such as those found in [37][101]. Our goal is to simulate these complex healthcare scenarios by way of decentralized and policy-compliant COAST-based services. Through these simulations we determine if our proposed techniques enable diverse parties to access and manipulate capability and information based on provider-defined and trust-based authorizations, therefore answering our research question (section 3).

More concretely, we select scenarios which reflect the goals set forth in section 3, namely:

1. G1: Enable fine-grained control over the access to a provider's services (both data and

computation capabilities) by supporting differential access according to specific trust and legal relationships.

2. G2: Allow a service provider to revoke the capabilities granted to service consumers according to changing trust relationships.

3. G3: Enable service composition and customization that allows the user to fulfill specific needs.

4. G4: Enable integration of information from different sources under different spans of authority.

Therefore, across vignettes we expect to address the provider's ability to capture relevant information access policies and create policy-conforming services; the service consumer's ability to customize a service—composed from the functions offered by a binding environment—to suit specific needs; the ability to use CURLs to enforce differential access to services; and the effort involved in integrating information from different sources.

We then use COASTmed as the artifact for empirical evaluation to build a set of simulations based on these selected scenarios. These scenarios and corresponding simulations are thus dispositive in demonstrating that these goals can be achieved with our techniques, namely that users can use and customize services they are entitled to, but cannot perform computations that violate organizational privacy policies. Scenario-based evaluations have been also adopted in related work such as the Cassandra trust management system [37].

We make assumptions regarding trust and reputation management conducted by participating parties as discussed in section 6.3 and address the enactment of these scenarios not from an end-user/user interface perspective, but from an underlying system development one.

8.1.1 Simulation Setup

To run our simulations, we set up an arrange of islands corresponding to service users and providers within the evaluation scenarios. Recognized system users have a unique identifier, which is a number for purpose of simulation, but meant to be a unique identifier such as a public key. How providers obtained these unique identifiers is out of the scope of our research, but we assume they are obtained through Web of Trust mechanisms or by way of certified credentials.

To better visualize the inputs and outputs of the simulations, we have built a simple user interface, which allows requesting a service CURL for a specific user (UUID), viewing the bestowed capabilities for a given user, and using the provided capabilities by way of user-created custom closures (figure 8.1). The providers' public CURLs to which a user sends a message to obtain a service CURL is hardcoded in the client for convenience. However, a public CURL is meant to be obtained as a serialized text file by email or by downloading it from the provider's website [123].

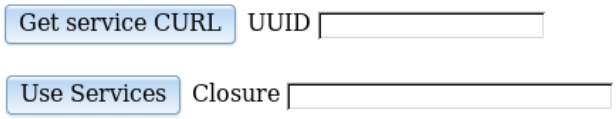

Figure 8.1: Simulation's main screen.

We implemented dummy capabilities for the purpose of demonstration. Domain facts required for the evaluation of policies are obtained from a backend database, whose schema mostly resembles the Prima Care open source EHR system (see section C). In addition, we will refer across scenarios to a set of hypothetical users, whose names, roles, and unique identifiers are described in table 8.1.

UUID	Name	Role
1	Jim	physician
2	Jackie	nurse
3	Tom	intern
4	Laura	administrator
5	John	patient
6	Jane	patient
7	CDC	government
8	Mary	resident
9	Peter	physician

Table 8.1: Sample users

8.1.2 Goal 1: Differential Access Support

Objective: Demonstrate how fine-grained control over the access to a provider's services (both data and computation capabilities) is enabled by supporting differential access according to specific trust and legal relationships.

Scenario 1a: Hafner et al. present a scenario where a physician can view and modify any medical record for which he or she is the designated primary physician [129]. The context of their work is the UCON policy model and the SECTET framework for model-driven security for healthcare architectures. This scenario illustrates information access conditions based on role and relationship with a third party.

Formal policies: the policies which capture the physicians' rights with respect to their patients' EHR, namely their ability to view and update a patient's EHR on the condition that the physician is the patient's primary care one are:

```
policy(physician,EHR/view,patient,null,(primaryPhysician(*
    physician*,*patient*)))
policy(physician,EHR/update,patient,null,(primaryPhysician(*
    physician*,*patient*)))
```

Simulation setup: For this simulation we have four principals, a hospital holding electronic health records, Jim who is a physician (UUID 1), and patients John (UUID 5) and Jane (UUID 6). Jim is John's primary care physician, but not Jane's. There are two islands in this simulation: island $I\ EHR$ with address www.EHR.com:5000 managed by the hospital and island $I\ Jim$ with address www.JIM.com:5001 running on Jim's PDA.

Inputs and expected outputs: Jim obtains a service CURL issued by $I\ EHR$ bestowing him capabilities `EHR/view` and `EHR/update`. He first sends a closure `lambda()(EHR/view 5))` whose goal is to view John's EHR. The expected output of such computation is John's EHR. Since we are implementing dummy functions, the service simply returns a notification string which confirms the access to the EHR, but in reality a vector of values, a string, a hash map, or XML-formatted information as defined by the user with health information is meant to be returned.

As a counterexample, Jim sends a a second message with the closure `lambda()(EHR/update 6))` whose goal is to update Jane's EHR. Since the policy states that a physician can update an EHR only if the physician is the patient's primary care doctor, we expect that the access to this record will be denied since Jim is not Jane's primary care physician.

Simulation result: the simulation run with our prototype demonstrates the expected outputs. Figure 8.2 shows Jim's (UUID 1) authorized capabilities (`EHR/view` and `EHR/update`) as well as the result of message `lambda()(EHR/view 5))` sent to the provider ($I\ EHR$). Likewise, figure 8.3 shows the expected output, where Jim (UUID 1) is forbidden from accessing Jane's EHR (UUID 6). The previously described policies, embedded in the CURL issued to Jim, are evaluated upon message receipt (as described in section 7.5.1). The provider's fact database, in the former case, confirms that Jim is John's primary care physician so the policy `policy(physician,right(EHR/view,(patient),(primaryPhysician(*physician*,*patient*))))` evaluates to *true*. In the later case, policy `policy(physician,right(EHR/update,(patient),(primaryPhysician(*physician*,*patient*))))` eval-

uates to *false*, since Jane is not Jim's patient.

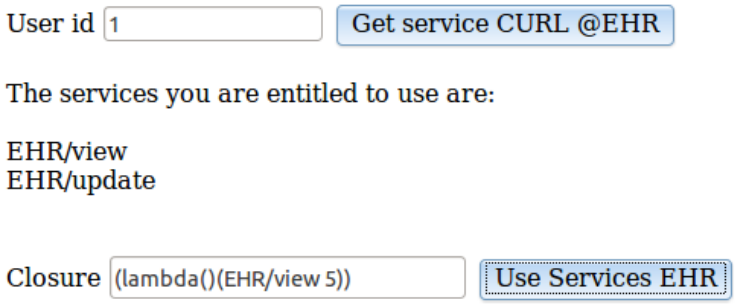

Figure 8.2: Scenario 1a: Jim (UUID 1) viewing John's EHR (UUID 5)

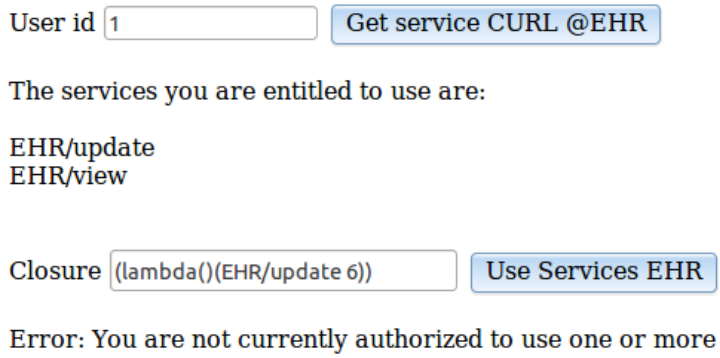

Figure 8.3: Scenario 1a counterexample: Jim (UUID 1) attempting to update Jane's EHR (UUID 6)

Scenario 1b: Ferreira et al. describe a scenario where nurses must have read access to the EHR of the patients registered within their department [106]. Their work focuses on "break the glass" policies within a virtual electronic medical record system. The conditions for service access in this scenario is not only role but affiliation.

Formal policies: the following policy, relevant to this scenario, allows a nurse to view an EHR if the patient is registered in the same department to which the nurse is affiliated.

```
policy(nurse,EHR/view,patient,null,(equal(employeeDepartment(*
    nurse*), patientDepartment(*patient*)))
```

Simulation setup: in this scenario, there are four principals involved—the hospital, Jackie who is a nurse within the Intensive Care Unit (UUID 2), and patients John (UUID 5) and Jane (UUID 6). John has been admitted to the Intensive Care Unit, while Jane is being treated in the Ambulatory Unit. Two islands interact in this scenario: island $I\ EHR$ with address www.EHR.com:5000 managed by the hospital and island $I\ Jackie$ with address www.JACKIE.com:5002.

Inputs and expected outputs: in this simulation, we expect Jackie to only be able to access the capability `EHR/view 5` as described by the policy. It is expected that Jackie can view John's EHR by sending a message `lambda()(EHR/view 5))` to $I\ EHR$ since both Jackie and John are registered as nurse and patient within the ICU. Instead, we do not expect Jackie to be able to access Jane's EHR, since Jane is in ambulatory care.

Simulation result: figures 8.4 and 8.5 show the inputs and outputs of this scenario. Jackie obtains the service CURL issued by $I\ EHR$ which entitles her to read health records. The result to message `lambda()(EHR/view 5))` is John's EHR, while the result of message `lambda()(EHR/view 6))` is an error message, since Jackie is not entitled to access Jane's EHR according to the policy. To determine this authorization, the condition `equal (employeeDepartment(*nurse*), patientDepartment(*patient*))` is executed; in

the first case `equal(employeeDepartment(2), patientDepartment(5))` returns *true*, while in the counterexample case `equal(employeeDepartment(2), patientDepartment(6))` returns *false*.

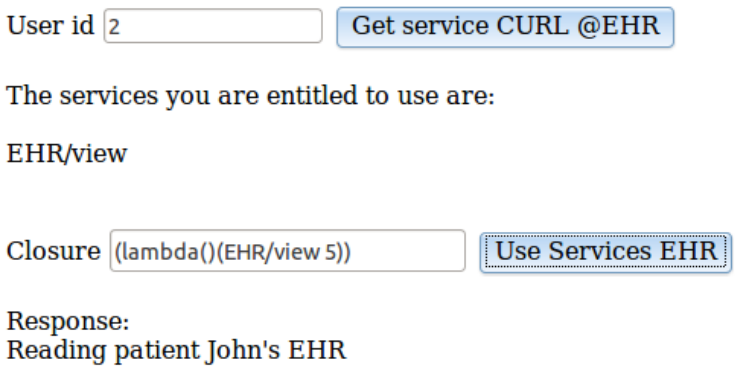

Figure 8.4: Scenario 1b: Jackie (UUID 2) viewing John's EHR (UUID 5)

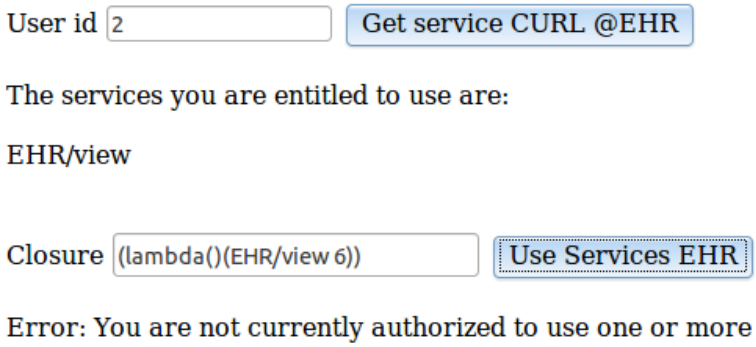

Figure 8.5: Scenario 1b counterexample: Jackie (UUID 2) attempting to view Jane's EHR (UUID 6)

Scenario 1c: The following scenario is based to the one found in Bhatti et al., who integrate WS-Policy with X-GTRBAC to bridge the gap between web services and access control policies [47]: interns may access health records between 9AM and 5PM. This scenario illustrates contextual access rights according to environment conditions.

Formal policies: the following policy states that interns are allowed to view any EHR as long as it is between 9AM and 5PM.

```
policy(intern,EHR/view,null,null,(after(*currentTime*, 9AM),
    before(*currentTime*, 5PM))))
```

Simulation setup: three principals are involved in this scenario—the hospital (service provider), Tom who is an intern (UUID 3), and the patient John (UUID 5). Two islands interact in this scenario: island $I\ EHR$ with address www.EHR.com:5000 managed by the hospital and island $I\ Tom$ with address www.TOM.com:5003.

Inputs and expected outputs: Tom as an intern has the right to view patient electronic health records. Therefore, we expect Tom to be able to access John's EHR between the hours of 9AM and 5PM, but not outside this time frame.

Simulation result: through the simulation, we can successfully demonstrate that Tom can only access John's EHR within the indicated hours (figure 8.6), but not after 5PM and before 9AM of the following day (figure 8.7). The policy variable `*currentTime*` is replaced by the current time according to the provider's system, and accordingly, the relevant policy evaluates to either *true* or *false*.

These three simulations demonstrate that we can achieve the goal of enabling the provider to exercise fine-grained access control to its services by defining formal policies which restrict access to capabilities according to role, target of the action, and external conditions.

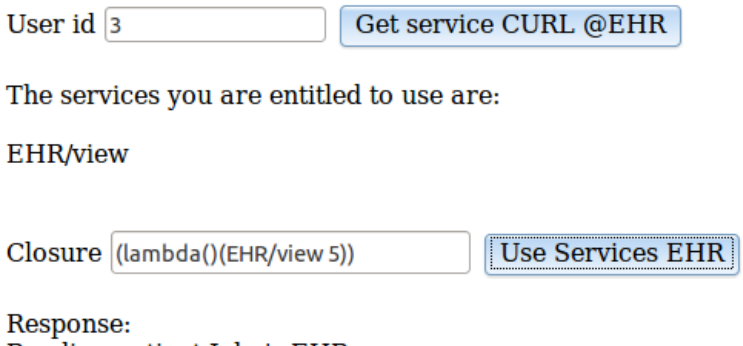

Figure 8.6: Scenario 1c: Tom (UUID 3) viewing John's EHR (UUID 5) sometime between 9AM and 5PM.

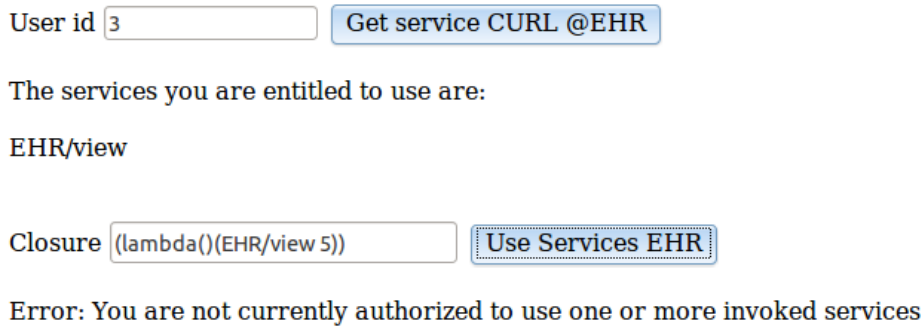

Figure 8.7: Scenario 1c counterexample: Tom (UUID 3) is denied to access John's EHR (UUID 5) after 5PM.

8.1.3 Goal 2: Capability Revocation

Objective: Demonstrate what service providers are able to revoke the capabilities granted to service consumers according to changing trust relationships.

Scenario 2a: In another scenario described by Ferreira et. al administrative staff have no access to the system at present [106]. We assume in this scenario that an administrator has at t_0 access to view hospital employees' payroll, and at t_1 this access right was revoked. With this scenario we illustrate a changing relationship between service user and provider, where the provider revokes completely the access previously granted to a service.

Formal policies: the service provider defines a policy which states that an administrator has the right to view the organization's payroll information.

```
policy(administrator,PAYROLLS/view,null,null,null))
```

Simulation setup: there are only two participants in this simulation, the hospital providing the information service and Laura, a hospital administrator (UUID 4). Corresponding to these parties, there are two islands—$I\ EHR$ with address www.EHR.com:5000 managed by the hospital and $I\ Laura$ with address www.LAURA.com:5004.

Inputs and expected outputs: what we intend to demonstrate in this simulation is that at t_0 Laura is able to use a `view-payrolls` function by sending a message `(lambda() (PAYROLLS/view))`. However, at t_1 the provider revokes Laura's CURL and she can no longer access any service.

Simulation result: we run the simulation twice; on the first one, we observed the service providing Laura with the expected output (figure 8.8). Figure 8.9 instead, shows Laura's client obtaining an error message after the provider revoked the CURL (by setting the *revoked* field in the CURL registry database table to *true*).

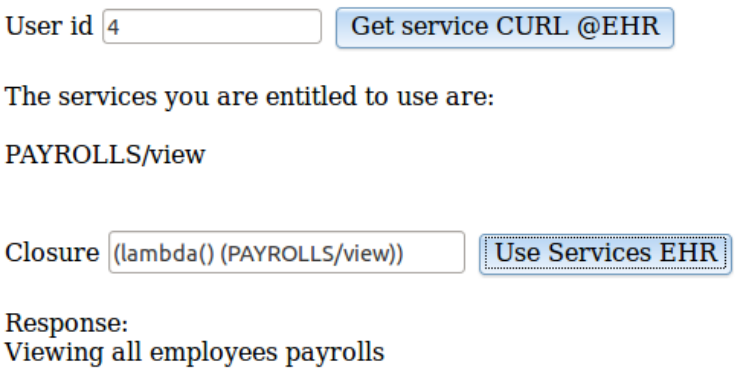

Figure 8.8: Scenario 2a: Laura (UUID 4), a hospital administrator viewing the organization's payrolls at t_0.

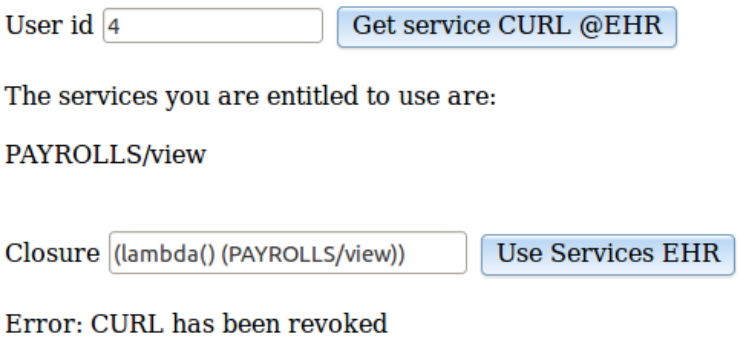

Figure 8.9: Scenario 2a counterexample: Laura (UUID 4), at t_1, receives an error message when attempting to use again the view-payrolls capability.

Scenario 2b: Gerraoui et al., in the context of the OASIS role-based access control model, describe a scenario where, before returning data to a particular healthcare provider, it is checked that the healthcare provider has not been excluded from access by the patient. This scenario is different than the previous one in that it is not the provider revoking a capability to a service, but where the patient forbids the access to his/her record to a healthcare provider.

Formal policies: the policies that are relevant in this scenario are, first, a policy which states that a resident has the right to view an EHR unless it is a forbidden EHR. A second policy instead allows patients to forbid the access to their health record to particular resident.

```
policy(resident,EHR/view,patient,null,(not(forbiddenEHR(*resident
    *,*patient*))))


policy(patient,EHR/forbid,person,null, null)
```

Simulation setup: in this simulation there are three principals, the hospital providing the information service, Mary the resident (UUID 8), and John the patient (UUID 5). There are three islands in this simulation: island $I\ EHR$ with address www.EHR.com:5000 managed by the hospital, island $I\ Mary$ with address www.MARY.com:5008, and $I\ John$ with address www.JOHN.com:5005.

Inputs and expected outputs: First, we expect for Mary to be entitled to view and update patients EHRs. He sends a closure `lambda()(EHR/update 5))` which returns John's EHR. Then, John, using his own user interface within a mobile phone, forbids the access to John to view his EHR sending message `(lambda() (EHR/forbid 8 5))` to the hospital's service. We expect that, after the user executes this closure in the service, Mary will not be able to access John's EHR again.

Simulation result: COASTmed behaved as we expected. Figure 8.10 shows Mary successfully accessing John's EHR. Following, John forbids the access to his records to John as shown in figure 8.11. When Mary attempts to access John's EHR again, she receives an error message stating that the access to this patient's EHR is forbidden as shown in figure 8.12.

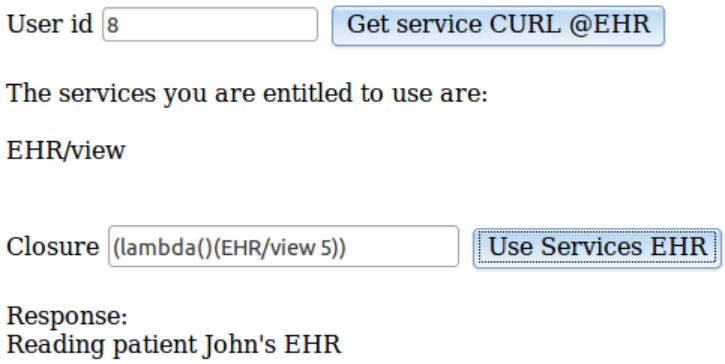

Figure 8.10: Scenario 2b t_0: Mary (UUID 8), a hospital resident accessing John's EHR (UUID 5).

Scenario 2c: In addition, Gerraoui et. al describe a scenario where access is not permitted to a specific principal despite of their role: Jane's health record may not be accessed by Tom. This scenario is similar to the previous one, but illustrates how permissions and exception can be described in terms of a principal's identity rather than on his/her role.

Formal policies: the following policy states that an intern can view a health record unless the patient is Jane and the intern is Tom (recall we cannot express prohibitive policies, but we can express exceptions to a rule).

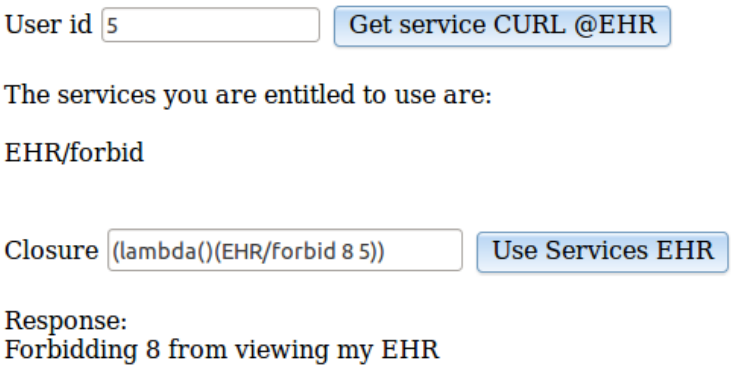

Figure 8.11: Scenario 2b t_1: John (UUID 5) forbids Mary (UUID 8) from accessing his EHR.

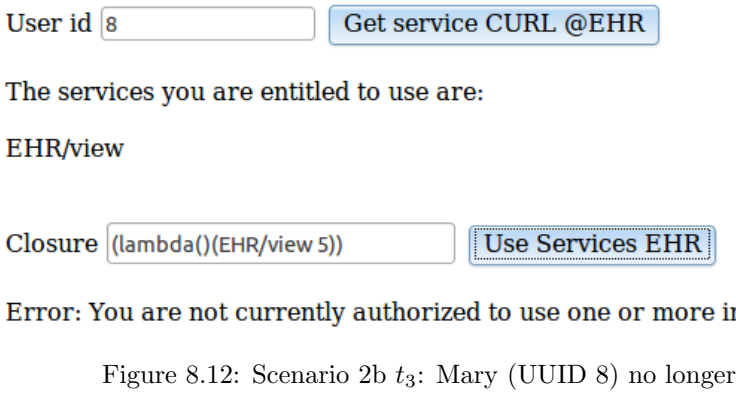

Figure 8.12: Scenario 2b t_3: Mary (UUID 8) no longer can access John's EHR (UUID 5).

```
policy(3,EHR/view,patient,null,(not(equal(*patient*,6))))
```

Simulation setup: three principals, the hospital, Tom the intern (UUID 3), and Jane the patient (UUID 6) are involved in this scenario. There are two islands in this simulation: island $I\ EHR$ with address www.EHR.com:5000 managed by the hospital and island $I\ Tom$ with address www.TOM.com:5003.

Inputs and expected outputs: we expect that when Tom obtains a service CURL, the granted capability to be `EHR/view`. However, when he sends a closure `(lambda()(EHR/view 6))` to the hospital service, access should be denied since the condition `not(equal(*patient*,6))` should return true. In this case, the relevant policy is selected based on user's identity

in addition to relevant role-based policies for Tom.

Simulation result: figure 8.13 shows the expected result, where Tom (UUID 3) is denied from accessing Jane's EHR (UUID 6). As a counterexample, figure 8.14 shows Tom successfully accessing John's EHR (UUID 5).

This set of scenarios have demonstrated that a provider can completely revoke the access previously provided to a principal, a principal can be excluded of accessing specific information, and that access to a capability or to perform an action on a particular target can be denied on the basis of both user and target identities.

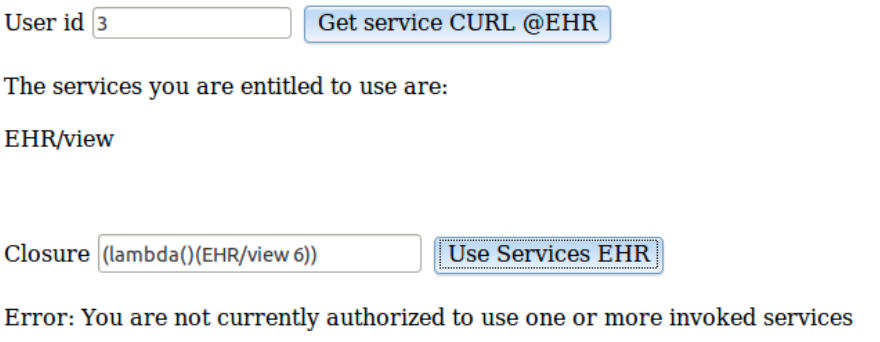

Figure 8.13: Scenario 2c: Tom (UUID 3) is denied the access to Jane's EHR (UUID 6).

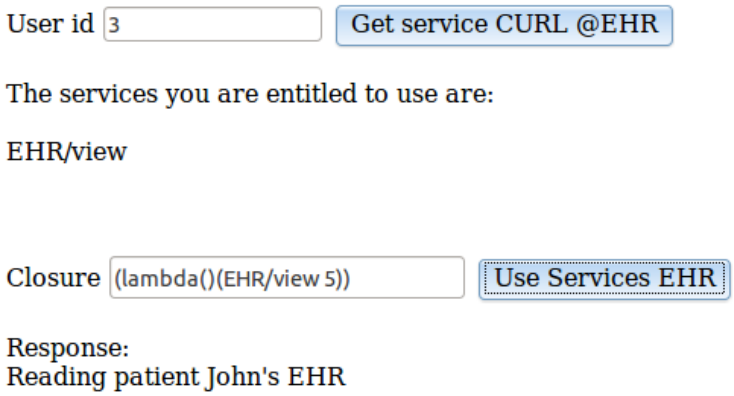

Figure 8.14: Scenario 2c counterexample: Tom (UUID 3) can successfully access John's EHR (UUID 5).

8.1.4 Goal 3: Service Customization

Objective: Demonstrate that it is possible to enable service composition and customization by using the capabilities of one or more providers allows users to fulfill specific needs.

Scenario 3a: Sobolev et. al describe a scenario where a patient is issued a referral to a specialist, which the patient uses to schedule an appointment [238]. Their work includes Statecharts, a graphical specification system for representing reactive systems as and extended version of finite-state machines that consider parallelism and event broadcasting. This scenario illustrates how a user can compose custom closures that leverage capabilities from disparate sources to fulfill specific needs. The communication pattern this scenario follows is that of a dependent star (section 6.2.2).

Formal policies: two different providers independently define access control policies. The hospital defines a policy

```
policy(patient,REFERRAL/get,specialist,null,null)
```

which enables a patient to obtain a referral. A second provider corresponding to the specialist physician defines, in an scheduling system, a policy

```
policy(patient,APPT/schedule,patient,null,null)
```

which allows a patient to schedule an appointment.

Simulation setup: three principals, the hospital and the specialist (both service providers), and Jane the patient (UUID 6) are involved in this scenario. There are three islands in this simulation: island $I\ EHR$ with address www.EHR.com:5000 managed by the hospital and, $I\ SPEC$ with address www.SPEC.com:5008 managed by the specialist, and island $I\ Jane$ with address www.JANE.com:5006.

Inputs and expected outputs: in this simulation we expect the user to receive a appointment

confirmation as a result of a message

```
(new/motile/procedure
  `(lambda()
    (curl/send ,curl-specialist
      (service-call/new
        (new/motile/procedure
          `(lambda()
            (APPT/schedule ,(REFERRAL/get 9 6))))
      #f ,user))))
```

sent to @EHR which first requests a referral from $I\ EHR$ (by way of the expression `(REFERRAL/get 9 6)`) and then the computation proceeds (`(curl/send ,curl-specialist ....)`) with such a referral to make an appointment with the specialist (at $I\ SPEC$) (through the execution of expression `(APPT/schedule [result of,(REFERRAL/get 9 6)])`).

Simulation result: figure 8.15 shows the screenshot of the simulation, where, first, Jane clicks on both "Get service CURL @EHR" and "Get service CURL @SPEC" buttons to obtain the service CURLs from both providers. $I\ EHR$ grants her the `REFERRAL/get` capability, and $I\ SPEC$ grants her the `APPT/schedule` capability. For this simulation we built a slightly different user interface since we are building complex closures whose expressions are executed by different computations. When the user clicks on the "Send Complex Closure" button, the closure in the previously defined code snippet is sent to $I\ EHR$. The output of this computation is an appointment confirmation message sent by the specialist ($I\ SPEC$).

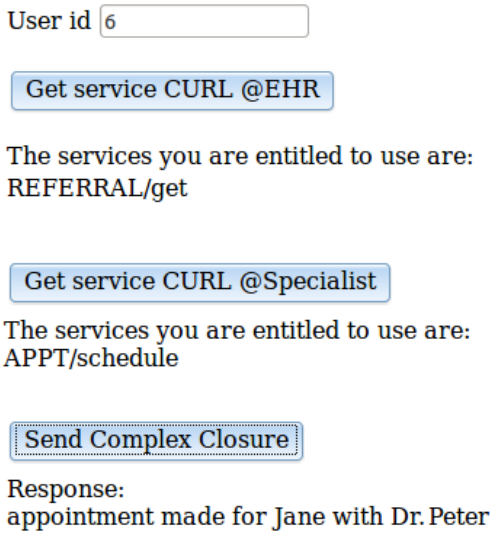

Figure 8.15: Scenario 3: Jane (UUID 6) composes a custom closure to obtain an specialist referral from the hospital an make with it an appointment with the specialist.

8.1.5 Goal 4: Data Integration

Goal: Enable integration of information from sources under different spans of authority.

Scenario 4a: Kukafka et al. discuss extending EHR systems to support public health concerns, leveraging individuals' data to make community-level assessments that can aid clinicians to diagnose and treat, researchers to recruit clinical trial volunteers, and policy makers [162]. One of the requirements is infectious disease surveillance based on real-time data collection in the face of low reporting compliance rate. Therefore, we can imagine a scenario where the Centers for Disease Control and Prevention obtains illness information from healthcare providers to detect in real-time disease epidemics and issue community alerts and prevention measures. With this scenario we show how a user can obtain information from different sources to thereafter process this information locally. The communication pattern that this interaction fulfill is that of an independent star (section 6.2.2).

Formal policies: in this scenario, a hospital providing services defines the policy

```
policy(government,EHR/viewAnonymous,null,null,(authorized(*
    government*)))
```

which states that only authorized government agencies may access anonymous health records. Another healthcare provider, a clinic, specifies also a policy that has the same semantics.

Simulation setup: in this scenario we have three principals, a hospital holding electronic health records, a clinic holding a separate set of health records, and the CDC (UUID 7), an organization interested in detecting epidemiological outbreaks. In this simulation there are three islands: island *I EHR* with address www.EHR.com:5000 managed by the hospital, island *I Clinic* with address www.CLINIC.com:5008 managed by a clinic, and island *I CDC* with address www.CDC.com:5007 run by the CDC.

Inputs and expected outputs: first, the CDC obtains service CURLs from both the hospital and the clinic by sending a message to their public CURLs @EHR and @Clinic correspondingly. They both bestow the capability `EHR/viewAnonymous` to the CDC. The CDC sends a custom closure

```
(lambda()
  (letrec
    ((EHR-list (EHR/anonymous)) ;;retrieves the anonymous EHR
    (hash2014 (make-hasheq)) ;; to store 2014 diagnosis
    (hash2013 (make-hasheq)) ;; to store 2013 diagnosis

     ;;function which returns increase in diagnostics per illness
    from 2013 to 2014 if it is more or equal to %50
    (increase-perc (lambda(key-list)
      (cond
        ((empty? key-list) null)
        ((<= 50 (/ (* 100 (- (hash-ref hash2014 (car key-list))
```

```
                    (hash-ref hash2013 (car key-list))))
                (hash-ref hash2013 (car key-list))))
          (cons (list (car key-list)
              (/ (* 100 (- (hash-ref hash2014 (car key-list))
                    (hash-ref hash2013 (car key-list))))
                (hash-ref hash2013 (car key-list))))
            (increase-perc (cdr key-list))))
          (else (increase-perc (crd key-list))))))

;; function to transform the list of illness increases to a string
    (toString(lambda(l s)
      (cond
        ((empty? l) s)
        (else (toString (cdr l) (string-append s (caar l) " " (
   number->string (cadar l)) "%<br />")))))))

;; aggregates the number of patients per illness, per year
  (map
    (lambda(patient)
      (let ((illness (vector-ref patient 1))
        (hash
          (cond
            ((equal? (vector-ref patient 2) 2013) hash2013)
            (else hash2014))))
      (cond
        ((hash-has-key? hash illness)
        (hash-set! hash illness (+ 1 (hash-ref hash illness))))
        (else (hash-set! hash illness 1)))))
    EHR-list)
```

```
;; returns the result as a string and the aggregated results for
  all illnesses for both 2013 and 2014
(list (toString (increase-perc (hash-keys hash2014)) "")
 hash2013 hash2014)))
```

which uses the provided anonymous health records to detect alarming increases in diagnostics from 2013 to 2014 for different illnesses.

Simulation result: figure 8.16 shows the expected results to the CDC's custom computations by retrieving the spikes in diagnosis for given illnesses. The CDC can then locally integrate this different information from decentralized healthcare providers since the number of diagnostics per illness, per year is also returned as part of the response. Therefore the CDC can perform the same infectious illness increase calculations at a regional or national level.

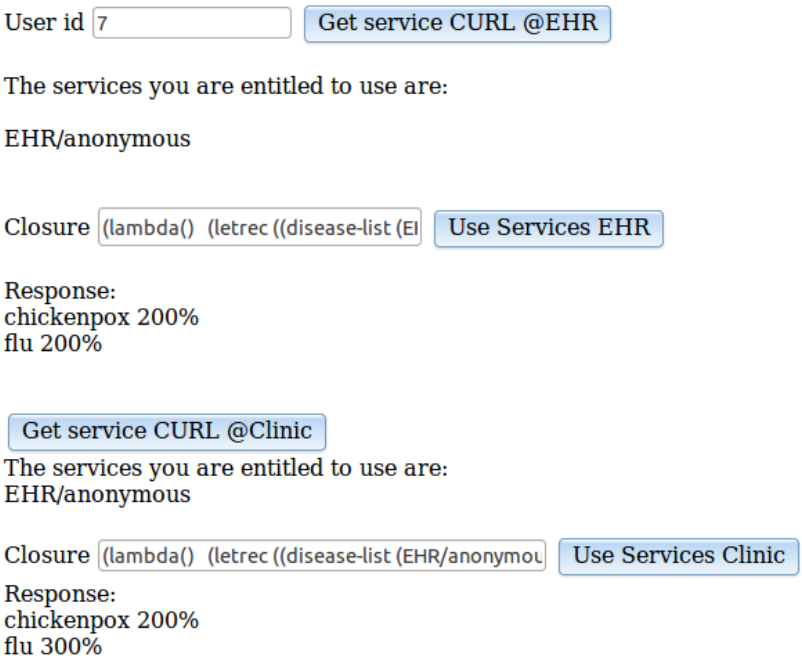

Figure 8.16: Goal 4, scenario 2 input and output for CDC.

Scenario 4b: In this scenario a patient consolidates his data from decentralized pharmacies, clinics, and hospitals. We illustrate how a computation can travel through a series of services aggregating information before the result is sent back to the user. This complex communication pattern is what we refer to as ring topology (section 6.2.2).

Formal policies: a hospital defines a policy

```
policy(person,EHR/view,patient,null,(equal(*person*,*patient*)))
```

which allows a patient to retrieve his/her EHR. At the same time, a clinic describes a similar policy. A third health services provider, namely a pharmacy defines a policy

```
policy(person,PRESCRIPTIONS/view,patient,null,(equal(*person*,*
    patient*)))
```

which allows a patient to retrieve his/her historical record of ordered prescriptions.

Simulation setup: in this scenario we have three service providers—a hospital, a clinic, and a pharmacy—and a service user, the patient John (UUID 5). There are four islands in this simulation: island *I EHR* with address www.EHR.com:5000, island *I Clinic* with address www.CLINIC.com:5008, island *I Pharmacy* with address www.PHARM.com:5008, and island *I John* with address www.JOHN.com:5005 managed by John.

Inputs and expected outputs: John is interested in aggregating to his local EHR his records held at different providers. He clicks on the three buttons "Get Service CURL @EHR", "Get Service CURL @Clinic", and "Get Service CURL @Pharmacy" to obtain the service CURL issued by each provider. We expect that given the stated policies the hospital, clinic, and pharmacy will provide him with the capabilities `EHR/view`, `EHR/view`, and `PRESCRIPTIONS/view` correspondingly. Then, John sends a complex closure

```
(new/motile/procedure
  `(lambda()
     ;; retrieves John's EHD from the hospital
    (let ((EHR-hospital (EHR/view 5)))
      ;; sends the closure to the clinic's service carrying with
   it the hospital's EHR
      (curl/send ,curl-clinic
          (service-call/new
            (new/motile/procedure
              `(lambda()
              ;; retrieves John's EHR from the hospital
                (let ((EHR-clinic (EHR/view 5)))
                  ;; sends the closure to the pharmacy's service
   carrying with it both the hospital's and the clinic's EHR
                (curl/send ,,curl-pharmacy
                  (service-call/new
                    (new/motile/procedure
                      `(lambda()
                        (let ((prescr (PRESCRIPTIONS/view 5)))
                          (cons ,,EHR-hospital
                            (cons ,EHR-clinic (list presc))))))
                  #f ,,user)))))
          #f #f)))))
```

to *I EHR* which collects John's EHR, then travels to *I Clinic* collecting John's record at the clinic, proceeds to *I Pharmacy* to retrieve John's prescription history information, and finally comes back as a response to John with all the aggregated records within a vector.

Simulation result: the result of this complex interaction can be observed in figure 8.17 which

shows the capabilities each provider bestows to John, the input with this complex closure, and the aggregated output—three strings which stand for John's EHR held at the hospital, his EHR held at the clinic, and the list of prescriptions gathered from the pharmacy.

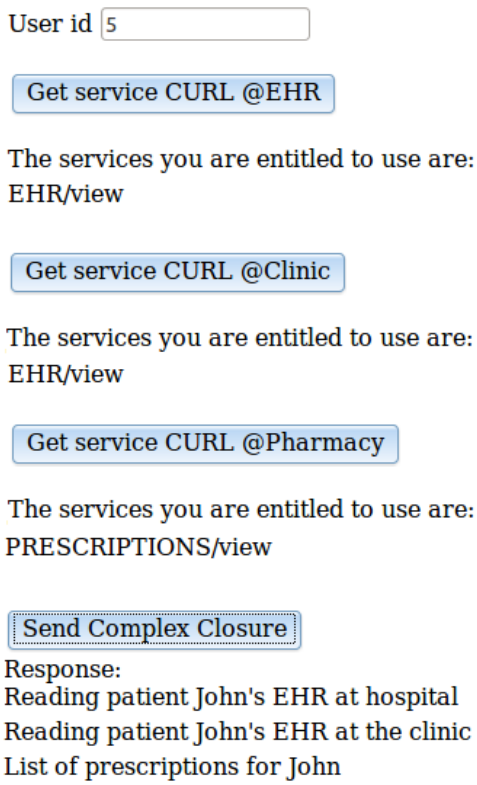

Figure 8.17: Goal 4, scenario 1 input and output for patient John.

8.2 Comparative Analysis

In this stage of the evaluation, we assess COASTmed and COAST-based services more generally through comparative analyses with respect to the desired properties with a representative set of technologies and systems approaching similar challenges regarding web services, privacy, access control, and customization. Most of them have been found in the authorization and trust management literature, which emphasizes the need for formalizing policies to support access control mechanisms. These are either healthcare specific or appropriate for the healthcare context. Specifically, we assess the ability of these technologies to (a) expressively capture privacy and operational policies; (b) offer policy compliant services; (c) provide user-specific services, and; (d) allow users to customize the service to fit individual requirements within the granted functionality. In doing so, we provide insight on how our approach is superior and not just different. This evaluation is limited to functional capabilities and excludes usability and user interface aspects.

Following, we provide an overview of the evaluated technologies and systems (section 8.2.1 and 8.2.2), and describe (section 8.2.3) and discuss (section 8.2.4) the results of our analysis.

8.2.1 Technologies Overview

SOA Orchestration and Choreography

Benyoucef et al. [41] set forth the criteria for effectively modeling healthcare processes and assess the suitability of leading SOA technologies—orchestration and choreography—with respect to these criteria. Their motivation is the need to formally capture complex healthcare processes given that physical processes are increasingly replaced by electronic ones. This is a challenging endeavor since healthcare processes are highly dynamic and process integration, collaboration, and exception handling need to be achieved among decentralized healthcare

information systems. The goal of this work is to support healthcare processes through automated technologies. More specifically, their focus is micro-level decentralized processes and data flows that relate to individual patients and their healthcare providers.

Orchestration is the executable process within or controlled by a single organization, while choreography is the sequence of inter-organizational messages that are part of a business process. Comparison criteria of languages for web service orchestration and choreography (i.e. BPEL and WS-CDL correspondingly) includes usability, capabilities, and evolution. Specific evaluation features are tool support, ease of use, scalability, ease of monitoring, abstraction, security, privacy, exception handling, peer-to-peer representation, human vs automated processes representation, reusability, and maintainability. The comparison of these technologies is done in the context of a scheduled workflow which involves canonical patient medical encounters, whose steps depend on temporal and sequential constraints (registration, ordering of diagnostic images, image acquisition, and examination and viewing).

Researchers found that neither orchestration nor choreography can fulfill the requirements set forth in their evaluation to model complex healthcare processes. Most relevant to our work is the evaluation of security and privacy features. Neither web service orchestration nor choreography satisfactorily support security and privacy of service assets. Assessments in this work show that orchestration technologies (BPEL in specific) cannot represent task- nor process-level security. BPEL can be extended through plug-ins to include security features, however at increased complexity. In addition, not every orchestration tool supports extensions. Likewise, the WS-CDL choreography language, although it is partially integrated with WS-Security, has no capabilities to represent secure communications which are of outmost importance between decentralized parties. In both cases, communication security and data integrity may be separately supported by interaction protocols.

Orchestration does not support organization nor patient privacy since the evaluated technology has no construct for modeling privacy. Also, information about internal, potentially

private processes need to be shared with the overall orchestration process, and thus it is visible to other parties—in BPEL, processes and data are stored in a single location that is the BPEL server; external technologies are required to secure BPEL processes. Choreography partially supports privacy since private processes and data management can be made invisible to other parties through abstract process interfaces, and expose only what is required to be known for process participation in a peer-to-peer style. The authors however agree that the choice between orchestration and choreography is contextual and a combination of technologies may compensate for each other weaknesses.

Comparing widely adopted SOA technologies for combining services such as the ones discussed in this work is natural since our work is concerned, from a users' perspective, with the integration of information from decentralized services. Although our work is amenable to the decentralized nature of service choreography, it is more related to orchestration—our goal is giving control to the provider to securely expose services and data which may involve an "orchestration" of internal services, and allow users to combine different services to obtain the required data. Therefore our focus is providing control and flexibility to individual users and providers to participate in decentralized processes. The great advantages of our work over these technologies are that the participation of services in a process can be differentially provided according to the user's identity. Also, unlike this technologies, we provide the semantics and capabilities for user-side service customization. Neither orchestration nor choreography technologies can model customized interactions with process services.

XACML

In section 5.2 we introduced the eXtensible Access Control Markup Language (XACML), a language to specify access control policies for web resources. However, XACML is also an access control and data-flow model [203]. The main components of this model are: (a) the Policy Administration Point (PAP) which manages the creation of policies; (b) the Policy

Enforcement Point (PEP) which performs access control; (c) the context handler which receives an AC decision request from the PEP and constructs a XACML context request; (d) a Policy Decision Point (PDP) which evaluates the pertinent policies and makes an authorization decision to send back to the PEP; (e) the Policy Information Point (PIP) which retrieves attribute values that are needed by the PDP to evaluate policies.

A nontrivial disadvantage of XACML is that it is limited to provide a "permit" or "deny" request decision, but it is not clear how these policies and decisions are bound to services. The OASIS standard [203] loosely states "if access is permitted, then the PEP permits access to the resource; otherwise, it denies access". In addition, Bhatti et al. argue that XACML does not provide a mechanism to bind individual service components to specific policies [47]. That said, XACML is a very well thought and robust access control model, whose elements can be implemented as COAST computations handling different concerns of policy specification, evaluation, and enforcement, however with the increased security the COAST infrastructure provides and the flexibility of service use through mobile code.

SAML

The Security Assertion Markup Language (SAML) is an OASIS XML-based standard which allows defining assertions regarding the identity of a user to authenticate and authorize access to web services [60]. However, as noted by Bhatti et al. [47], SAML was not designed to describe authorization policies on its own, but its purpose is to share identity information among people and organizations, and more specifically exchange security tokens (containing assertions) between an identity provider and a service provider. SAML provides a single sign-on alternative to commonly used site-specific user passwords. Although SAML is a security standard for web services, it is not a competing approach, but potentially a complementary one for implementing the authentication component which our approach does not address.

WS-Policy

WS-Policy is a language for describing policies for web services [29]. Security policies describe web services' properties such as the type of encryption used, capabilities, and requirements. In other words, policies are assertions which advertise the conditions under which a service is provided. WS-Policy complements WSDL service description by providing additional information that otherwise would be provided out-of-band. Service consumers need to understand the semantics of these policies to decide on whether to interact with the service and what it is required to do so.

WS-Policy was not designed to support access control. The semantics of these policies are related to requirements and capabilities in terms of security of communication (e.g., use of one or the other cryptographic protocol, language, message format), and are tangential to the goal of overseeing domain-specific data-disclosure. In addition, as notified in the WS-Policy specification, this language is not concerned with how policies are attached to a Web service. Furthermore, WS-Policy is not a standalone standard, but is meant to be used in conjunction with application-specific protocols and other WS* standards, therefore increasing the complexity of any solution. The COAST infrastructure "as is" handles the underlying communication protocols and message encryption, and CURL metadata can be leveraged to advertise the accepted message formats (e.g., message types).

WS-Security

WS-Security [195] is a specification to augment SOAP-based communication with security mechanisms such as encryption, message authentication, integrity and confidentiality, user and password, SAML assertions, digital signatures, Kerberos tickets, X.509 certificates, and the like. With WS-Security, a SOAP message can be partially or completely secured through encryption, digital signatures, and security credentials [47].

The distinction between WS-Policy and WS-Security is that while WS-Policy advertises the requirements for communication, such as the type of encryption that ought to be used when communicating with a service, WS-Security provides the basis to implement it. WS-Security has the specific and narrow scope of addressing communication security, but these semantics are very different from access control and authority concerns.

Traditional Access Control

Mandatory, discretionary, role, and attribute-based are traditional access control models [175][275][103] used not only in the web services context, but more broadly in centralized and distributed systems where users have different rights with respect to system capabilities. Mandatory access control, mostly used in the military domain, assigns objects and subjects to labelled security classes. Labels describe the sensitivity of an object, while for subjects they describe the security clearance level. A subject can access an object if his/her clearance level is equal or higher to the one required by the object.

Discretionary access control (DAC) is also based on a collection of subjects and objects, a set of subjects' access rights with respect to specific objects, and optional constraint predicates. These rights are commonly read, write, and execute. For each object a record is maintained regarding the access rights that different principals have with respect to that object.

Role-based access control (RBAC) is perhaps the most common access control model. RBAC generalizes DAC by assigning rights to user roles as opposed to principals. These assignments are made at a system wide level, rather than being granted by the individual object owners. Namely, users do not have "discretionary" access to objects, but permission are assigned to roles, which have different authorization levels with respect to a system's objects. Principals can then access authorized objects by being assigned one or more roles. RBAC balances granularity and generalizability [47]. The rational behind RBAC is that

roles bestow responsibilities and qualifications that enable accessing certain resources and is a more scalable approach than DAC.

Lastly, attribute-based access control (ABAC) to objects is provided on the basis of subjects', objects', and environment's attributes. An access control rule specifies the combination of this attributes that must be fulfilled in order for a subject to be granted access to an object.

These access control models are not competing approaches to COAST-based services. In fact, COASTmed implements both RBAC and ABAC—policies may allow diverse access capabilities based on role membership, contextual conditions, and principals' and targets' attributes. COAST is silent on the conditions under which access to capabilities is bestowed; it is assumed that a CURL is issued to a trusted principal in order to use a set of capabilities. However, how trust is established and how of capabilities are assigned to each principal is specific to the application. This is why our approach involves complementing COAST with formal policies to capture access control semantics supported by these models.

However, these access control models on their own are not a solution to the problems approached—they are abstract, conceptual models, detached from implementation details. They neither provide a policy vocabulary nor specify how access control rules are effectively bounded to services. It seems that, the role of these models, in general, ends after a "permit" or "deny" decision has been made, and thereafter the security of the resource (and how to maintain this security) is left to the application programmer. In addition, they are focused on allowing providers to protect their assets, but have no regard on the users' needs, let alone provide customization capabilities (although it can be argued that personalization and customization can be provided on the basis of these models).

8.2.2 Systems Overview

AquaLogic Data Services Platform (ALDSP)

Borkar et al. present the access control mechanisms of the AquaLogic Data Services Platform (ALDSP), a middleware technology which allows building data services that integrate information from multiple sources [58]. A data service, as described by the authors, is an XML schema describing information along with a set of XQuery functions to access and manipulate such information. ALDSP can automatically generate data services from WSDL descriptions or SQL metadata. Clients use services by invoking their XQuery functions.

ALDSP is characterized by its focus on very granular user access control to resources including both business objects and relational database information. For instance, a resource hierarchy includes a data space, a set of data services, the elements or "shape" of individual services (i.e. of business objects), and accessor functions to the service elements. Access control policies can be applied at different levels within this hierarchy: at the data service level; at the function level by determining which users are allowed to invoke it and under what conditions, and; at the object element level by performing different actions with respect to the access to "securable" elements such as omitting, replacing, or encrypting information according to access rights. To manage these security configurations, ALDSP provides a user interface which allows selecting the "securable" elements and functions in the resource hierarchy, adding policy conditions, and linking XQuery security functions to secured elements. Policy conditions such as "access occurs after" or "deny access to everyone" are pre-defined and can be selected through this user interface. Alike in our approach, a user needs to meet the policy conditions that are applicable to the resource accessed through the data service. A subtle difference is that in COAST services access permissions are related to specific functions or actions, while ALDSP takes a more object oriented approach where restrictions are imposed on resources.

This approach involves a three-tier architecture, where ALDSP defines the data service elements and functions that are subject to access control (i.e. the context in which access control is needed), the underlying Web Logic Server (WLS) security framework manages and enforces policies, and an authorization provider (such as an XACML authorizer) performs accessibility checks based on the resource identifier, the user's identity, and any relevant context information; access to a resource is permitted or blocked accordingly. Also, ALDSP allows specifying additional XQuery security functions describing more complex access control rules for accessing a secured data element when it cannot be captured through WLS policies. Lastly, ALDSP can be integrated with the AquaLogic Enterprise Security (ALES) system for more advanced access control involving function replacement (when the user is not allowed to invoke a given function) and security filtering of retrieved data before it is sent to the user. Each of these components approaches a particular security concern, however at increased complexity in terms of the required component integration and the know-how required for policy makers to specify complex policies using all these diverse technologies. In other words, there is a trade-off between a clean separation of concerns through well defined components and having to manage different, independent technologies for the specification of security policies (i.e. XQuery functions, ALDSP resource hierarchy selection, WLS, and ALES). As a result, policies are scattered among the different layers of ALDSP's architecture. In addition, ALDSP does not provide an explicit policy formalism, but policies are captured by and visible through multiple graphic user interfaces provided by these technologies.

An important feature of ALDSP is considering service composition by allowing imposition of access control despite on whether the data service is being directly invoked or if it is part of a composite service calling. In other words, ALDSP considers the access control policies of all of a service's constituents. Indirect function invocation is discovered via inlining, which is in some way similar to our approach where mobile code sent by the user is inspected to perform access control based on pending policies. Another approached concern, which we have not considered in our studies, is optimization in the face of the overhead that access

control imposes on an application.

A very important distinction between this work an ours is that the assumption in ALDSP is that data services are public unless specific restrictions are placed on a resource hierarchy. Instead, our approach makes a closed world assumption, where everything is prohibited unless explicitly authorized. We believe that the principle of least privilege is far more secure in the face of potential policy maker errors and security oversights.

AquaLogic Data Services Platform (ALDSP) [58]
Problem addressed Fine-grained access control to data services.
Goal Ability to impose granular access control restrictions through data-driven security policies in SOA data services.
Contribution A middleware platform—with fine-grained access control capabilities—for building data services that integrate information from heterogeneous sources.
Scope data service architect or administrator
Architecture type Layered architecture
Prototype ALDSP which exhibits access control features such as the specification of data-driven security policies.
Used technologies WebLogic Server, XQuery
Domain Domain independent technologies, however appropriate for business processes.

Table 8.2: ALDSP summary

A Policy-Based Authorization Framework for Web Services (X-GTRBAC + WS-Policy)

Bhatti et al. [47] address a problem we also approach, namely the lack of differentiation web services make between users, making only binary access control decisions: authorized or non-authorized. Their approach to this problem is a policy-based authorization framework, where, alike our approach, formal policies are the basis for service use.

Policies—contextual constraints on service use—are specified in the WS-Policy profile of the

X-GTRBAC access control policy specification language. The integration of WS-Policy and X-GTRBAC bridges the gap between web services security and traditional access control mechanisms. In X-GTRBAC, permissions are based on roles; X-GTRBAC constraint expressions are used to impose rules for assigning users to roles and permissions to roles. A permission gives access to a specific service instance addressed by the "name" field within a WSDL specification. Policies can be associated individually to web service description (WSDL) components through policy attachments. One of their contributions is an algorithm to compute the effective access policy for a web service that is available to a role based on the relevant individual policy attachments.

Policy conditions may include temporal constraints and non-temporal logical expressions. These expressions capture rules related to subject roles' attributes. Constraints are satisfied if all conditions are satisfied.

To invoke a service, this must be registered and its usage policies published (in WS-Policy); policies are bound to WSDL component definitions. The user application invokes the service using the user's credentials. The Web application retrieves the service's policy attachments and merges them into a single use policy. The Web application then encodes the user's access request along with credentials within a SAML Authorization Decision Query and sends it to the PDP. The X-GTRBAC processor then evaluates assertions, comparing SAML information with WS-Policy specification and returns a SAML Authorization Decision Statement providing a "permit" or "deny" decision. Accordingly, the service is either invoked or not. A user can access a service if role membership and contextual constraints such as user location and system load are satisfied according to the service policy. Their architecture is a decentralized one, where the policy decision point and the policy execution point are managed by independent entities. This work also supports user authentication through credential expressions within policies which define the credentials that ought to be presented by the user to be assigned a role defined within the service.

According to our evaluation criteria, this approach captures formal policies, produces policy compliant services, and offers role-based services. Policies are bound to services by binding WSDL descriptions to policy assertions or predicates. This approach is similar to ours in that the a policy for a given principal is the combination of individual policy scopes—a collection of subjects to which a policy applies to—which include that subject. In COASTmed, the policy document that is relevant to a principal is the collection of policies whose subject is the principal (by identity or by role).

The core difference with our approach is that in X-GTRBAC + WS-Policy roles are associated to services, where in COASTmed roles are associated to individual capabilities, where a service is a set of capabilities. Another distinction is that services assign users to roles according to credentials, where roles are a local concept and may not be shared among services. In other words, services may require different credentials to assign a user to the role "employee". In COASTmed, roles are organization- and system-wide understood, and capabilities are provided according to those roles. In COASTmed policies are dynamically selected, as opposed to statically attached to services, and evaluated per each user; accordingly, a user-specific service is built. To the extent that we understand the operationalization of this approach, providing differential access to a growing number of users, requiring increasing number of WSDL descriptions, to the point of being unmanageable. In addition, users have the burden to figure out what is the appropriate and authorized service to use. COASTmed takes a different approach by saying "tell me who you are and I will tell you everything you are entitled to with respect to this service/organization and make it accessible through a single CURL".

Also, in Bhatti et al. a policy is relevant to a particular service access (which may include a set of operations on different ports), where for us, policies are particular to individual functions, the primitives of a service. Granularity is achieved through the exponential combination of roles and Web service instances. However, our COAST-based techniques provide far more

service granularity and flexibility—while in traditional web services a user calls on a single operation, in COASTmed users have available the power of functional composition.

One of the drawbacks of this approach is reliance on mapping two policy languages—X-GTRBAC and WS-Policy—adding to the complexity of policy specification. Also, no concrete policy example is provided by the authors, so it is not evident how to specify these policies and in what does mapping between the two languages consist.

X-GTRBAC + WS-Policy [47]
Problem addressed "Current authentication mechanisms for Web services do not differentiate between users in terms of fine-grained access privileges."
Goal Fine-grained, role-based access control to web services.
Contribution A policy-based authorization framework for web services.
Scope web service providers and consumer applications
Architecture type Service Oriented Architecture
Prototype Health information web service for physicians with varying authorizations.
Used technologies X-GTRBAC AC language and WS-Policy language. A Java-based policy decision point (PDP) and a PHP-based policy processing model.
Domain Any but example provided with relation to a health information system.

Table 8.3: X-GTRBAC + WS-Policy summary

Marmite

Mashups, web applications built by combining information and functionality from disparate sources, are an increasingly popular technology to manipulate information existing in distributed servers to fit information needs [274]. Data is asynchronously requested from distributed web services most often in an XML-based format or obtained through other methods such as web scraping and used as an application building block.

A disadvantage of these "end-user" approaches is that the user requires programming experience to create mashups. Although technologies such as Ajax, RESTful web services,

RSS, and Atom are rather simple and facilitate mashup development [274], mashups are not built by end users but by application developers. Addressing this disadvantage, the Marmite browser plugin provides a graphical interface for composing mashups [266].

Marmite follows a pipe and filter architecture, where three different types of operators—sources, processors, and sinks—manipulate data from remote sources. Sources gather data through web services or by navigating the HTML structure of web pages; processors alter in some way the data; sinks display the information by displaying it on a web page, saving it to a file, or other types of output. Data flows from sources to processors and finally to sinks.

Marmite and mashups more generally recognize that a provider cannot support the needs of all users, and therefore share our goal of allowing users to customize services. However, there are also fundamental differences with our approach. First, while mashup applications are composed by fixed, publicly available services that provide the same data to every user, our approach includes a more personalized access to data to which individual users are entitled to. Also, mashups and COAST-based services differ in the way service customization is operationalized; mashups rely on SOAP, RESTful web services, and Ajax, while custom services in COAST are achieved through mobile code. Our approach excels on its expressiveness to customize services through functional composition, not only being able to collect and manipulate information from a single source, but to compose computations that travel across multiple services to obtain the desired data in terms of content and presentation. Additionally, as argued by Yu et al. [274] mashups are an opportunistic use of services for short-lived applications, and are not meant for critical and business processes applications, for which our approach is fitting.

The advantage that Marmite offers is its usability and its reach to non-technical users by providing a user interface based on Apple's Automator. It provides simultaneous visualizations of the data operators through which data flows and of the resulting data. In comparison, our techniques, as they currently stand, require substantial programming skills. This ap-

parent disadvantage is an accidental and not an essential one; graphic tools can be built on top of the provided customization mechanisms to allow individuals to customize services in expressive ways. Our approach also assumes that organizations count with the required technical support and that application programmers build tools on behalf of end-user to access information.

Marmite [266]
Problem addressed Available web content is not in a form amenable to end users' needs.
Goal Allow end-users to create mashups to combine existing web content and services without requiring programing expertise.
Contribution The Marmite tool which allows end-users with create mashups from existing web content and services without requiring programming knowledge (Marmite offers a graphic user interface).
Scope End users
Architecture type Pipe and filter / data flow
Prototype Marmite, and end-user programming tool to create mashups.
Used technologies XUL, XPATH, Javascript, XBL.
Domain Any

Table 8.4: Marmite summary

Cassandra

Cassandra is an authorization and trust management system, which includes the policy language described in section 5.2. Cassandra focuses on trust management, thus enabling the establishment of trust between strangers—its role-based access control is contingent on the presentation of user credentials asserting user attributes. In other words, trust negotiation dictates who is authorized to request credentials and enables retrieving those credentials from some other party. Defined policies support automated credential retrieval—user credentials can be retrieved over the network. These credentials are issued by a registration authority and are the basis for access control. However, trust management is only conceptual, thus

has not been implemented in the presented prototype (it is meant to be implemented with digital certificates and public key signatures). Our approach also relies on trust mechanisms, however while their work is meant to rely on a certificate authority, COAST relies on self-certified islands along with decentralized Web of Trust (WoT) mechanisms.

Cassandra's policy specification semantics has strong focus on the activation and deactivation of roles and on specifying the entities that are authorized to activate system roles. There are four predefined operations in a Cassandra service: activating or deactivating a role, requesting a credential, or executing some action. COASTmed also supports role based-access control, but policies can be as low level as necessary, specifying per-individual or per-organization policies. Cassandra also considers the exchange and combination of inter-organizational policies, an aspect which our techniques does not address.

At a user's request, policies are evaluated to make an access control decision. Our approach is similar to Cassandra in this regard, however COASTmed evaluates some policies only once; others which depend on time or attributes of the specific service use instance are evaluated at every request. A substantial difference between Cassandra and our work it that, we explicitly bind policies with system bindings, therefore policies are effectively enforced. Cassandra makes an "accept" or "deny" access decision following policy evaluation, but it does not specify how services comply with policies from then on (i.e. how access rights are bound to capabilities).

Also, Cassandra is silent with regards to customization. However, we assume that it has the capability of personalizing services for users through policy-description-driven access control. However, services are still solely controlled by the provider. In contrast, our mobile-code-based solution provides the expressive power to allow users to customize a service in addition to the provider-controlled fine-grained access control to system functionalities.

Cassandra [37]
Problem addressed Specification and enforcement of security policies in large-scale distributed systems.
Goal Supporting distributed role-based access control based on digital credentials and formal authorization policies.
Contribution A policy language and a system to support role-based access control in distributed contexts.
Scope Service users and providers
Computing paradigm Functional paradigm for policy specification and imperative for the policy evaluator and the access control engine.
Prototype A prototype which includes capabilities to express and evaluate policies and an access control engine. Provides a user interface for reading policies and handling requests involving actions, credentials, and role activation.
Used technologies Datalog_C (Datalog with constraints) for policy specification and OCaml for prototype coding.
Domain Any, however presents evaluation scenarios in the EHR domain.

Table 8.5: Cassandra summary

Haas et al.

Haas et al. propose a privacy aware system where patients define privacy policies to control the disclosure of their data, held by specialized EHR storage enterprises, to third parties (however, it is not clear how policies are specified and whether they exist in natural language or expressed in some formalism). The important aspect of this work is that the patient does not need to trust the data holder, since this cannot access patient information unless it is authorized by the patient. This system also provides data disclosure logging so that patients can audit disclosures, detect policy violations, and trace the data legitimately disclosed. In addition, disclosed patient data is always accompanied with data provenance information (patient identity, data consumer, data provider, and relevant privacy policy). The enabling mechanisms are fingerprinting and digital watermarking to link data and provider and therefore legitimize of the disclosure. Consumers can be new data providers under the obligation of running a protocol to tag the data with new consumers' information. Through logging

and data tagging, data providers cannot repudiate unauthorized disclosures. A provided prototype includes the storage of patient data and the trust model based on privacy policies that enables information request access or denial, as well as enabling tracing data flows.

We share with this work the policy-based approach to the authorized disclosure of patient data. However, there are two fundamental differences: first, in Haas et al. the data holder has no authority over the held information; second, we consider that in COASTmed both the data holder and the patient could define privacy policies for the disclosure of patient data (although we have not implemented integration of patient policies yet). The advantage of Haas et al.'s work is the ability to tag data with provenance information, therefore not limiting policy compliance to a single interaction and disclosure, but addressing the continuity of compliance throughout the flow of patient data. However, their forensic approach to the disclosure through the identification of unauthorized disclosures, while valuable, does not prevent unauthorized disclosures despite the policies set forth; at this point any damage has already been done. Thus in practice policies are not associated to services. Given that policies are automatically bound to services in COASTmed, it is not possible to perform unauthorized disclosures to begin with. However, the limitation of our approach is that it does not consider how and to whom service consumers disclose the obtained service data.

InfoShare

The problem Jin et al. address is the frequent inability to make available, at the point of care and in a privacy preserving manner, patient medical history scattered across healthcare providers [143]. There is a need for access control frameworks to allow authorized users to retrieve health information from distributed EHR sources, integrating heterogeneous collections of data, and allowing patients to control the access to their data. Their goal is, therefore, to provide patients with a mechanism to authorize parties access to records held at specific data providers and to aggregated records created on the fly. The contribution

Haas et al. [128]
Problem addressed Electronic health records are stored by specialized enterprises which maintain this data. However, patients often have no control over the disclosure of their data.
Goal Protect patient privacy from unauthorized data disclosures by allowing patients to control and monitor the disclosure of their data.
Contribution A privacy-protecting EHR system with policy-based access control and data access logs, and a digital watermarking model to trace data flows through data provenance tagging.
Scope Patients and data providers
Prototype An electronic health record system which allows patients to control the disclosure of their health information through defined privacy policies.
Used technologies Asymmetric fingerprinting, digital watermarks, cryptographic commitments, authentication, data encryption.
Domain Electronic health records

Table 8.6: Haas et al. summary

is an access control model and policy scheme for sharing, at different levels of granularity, single EHR instances and aggregated data (considering different access control policies from each source) and resolves disparate and conflicting source's policies to aggregate EHRs.

EHRs are modeled as a property-labelled hierarchical structure. Labels assist in data filtering for authorized disclosure of data objects according to data properties. The authors present a prototype for sharing EHRs "views" through patient consent where the authorized data portion is matched with the user's request. Access to data is also conditional on usage purpose (payment, treatment, research, etc).

Our work shares the common goal of providing role- and policy-based access to private personal information. The fundamental difference is that while our work is more concerned with the data disclosure from a single data provider, this work emphasizes the aggregation of health data from different sources, combining policies not to violate individual data disclosure conditions. Policies are formally defined in a XPath-like notation such as

```
P4: (<GP,*>, ao, {treatment}, deny)
```

where `ao` is an object

```
(/VirtualEHR/History//*,<{h2},{general},*>)
```

in the EHR hierarchy. One of the disadvantages of this approach from a decentralization perspective is its dependence on a centralized registry service for patients and providers. Experience with SOA's UDDI suggests that service and data providers lack incentive to register in such registry services. It is not made clear, however, whether the service provider or InfoShare administrators are responsible for of registering a service to the registry.

Another disadvantage compared to our approach is complexity with regards to policy specification. Although they present algorithms to resolve conflicting authoritative and prohibitive policies, our approach is more simple by allowing only positive policies: everything that is not authorized within the policies is implicitly inaccessible.

Although the combination and aggregation of decentralized and potentially conflicting policies that Jin et al. address is a necessary and important contribution and future work we would also want to address, this is a concern that is subsequent to the one we approach. An advantage, however, of our work is concern for the utility of the service to the user, in specific service-consumer personalization and customization.

Privacy Policy Compliance System (PPCS)

This work [273] proposes an architecture for a Privacy Policy Compliance System (PPCS). Its design is informed by privacy legislation and requirements to protect personal privacy when using web services requiring personal data disclosure. Both consumer and providers have their individual privacy policies: consumers' policies describe the to whom, what, and in what circumstances information should be shared; providers state what personal information is required and how it will be used. If there is no conflict between consumer's and provider's

InfoShare [143]
Problem addressed Need to obtain, in a privacy-aware manner and according to corresponding authorizations, patient EHR information on the fly from multiple data providers.
Goal Guaranteeing that only authorized users can access EHR information from individual or multiple sources, considering aggregate and potentially conflicting privacy policies of individual data sources.
Contribution An access control scheme for selective sharing of patient information from multiple sources (virtual composite EHRs) and a policy scheme to resolve anomalies in aggregate inter-organizational policies.
Scope Patients, data providers, and healthcare providers
Computing paradigm Object oriented
Prototype InfoShare: an EHR sharing system for federated healthcare networks that provides authorized "views" of patient information to authorized parties. Patients define access control policies in the form of consent and based on these policies the system allows or denies user access. InfoShare retrieves and aggregates EHR data to compose a virtual composite EHR, resolving policy conflict in the process. Main architecture components are the Consent Management Service, Policy Composition Service, and EHR Authorization and Selection Service.
Used technologies HL7 Clinical Document Architecture (XML schema) for representing EHRs; XPath-like expressions to select EHR data elements; Jaxe XML editor for labeling data with properties; X.509 attribute certificates for patient consent; Java servlets for system implementation.
Domain Electronic health records

Table 8.7: InfoShare summary

policies, the provided web service is activated. In other words, the service's policies state "this is what I will do with your disclosed data; if you agree you may use the service, if not you cannot access it". The goal of this work if very similar to those of P3P platforms described in section 4.

The requirements that this architecture fulfills are accountability, identifying data collection and disclosure purposes, consent, limiting data collection, use, disclosure, and retention, data accuracy, security safeguards, openness on polices and practices, individuals' data access, and challenging policy compliance. A privacy policy specifies, for each data portion, a data collector, the nature of the data, purpose of collection, retention time, and the parties the

data may be disclosed to.

The proposed architecture includes three main components: a privacy controller, the Private Data Import/Export, and a set of Service Processes. The privacy controller monitors the flow of information between consumer and provider; makes log entries and allows accessing the log; deletes private information after use; allows consumer to update his/her personal information, and; informs consumers the disclosure of their information. The Private Data Import/Export component discloses and receives private data from other providers. Service Processes components are provider's services which collect data according to privacy policies.

One of the benefits of this work is that individuals can audit, by accessing logs, to whom and what data has been disclosed to third parties. A disadvantage, commonly found in our evaluation, is that privacy policies are seem to be loosely coupled with the behavior of system's services, providing opportunity for unauthorized disclosures to third parties. In addition, Yee et al. do not provide a concrete policy specification language, nor explain how conflicting policies between consumer and provider are resolved. They provide a conceptual architecture but do not include a proof-of-concept prototype, making the argument of the feasibility of this system weaker. Also, with respect to our research goals, this study does not consider neither service customization nor personalization.

Privacy Policy Compliance System (PPCS) [273]
Problem addressed Proliferation of web services requires protecting the privacy of web service users and that providers comply with privacy policies of users.
Goal Give service consumers control over their private information.
Contribution An architecture for privacy policy compliance systems.
Scope Information consumers and providers.
Domain Any

Table 8.8: Privacy Policy Compliance System (PPCS) summary

Web Service Access Control (Coetzee et al.)

This work provides a logic-based approach to access control for web services [76]. The authors propose extending WS-Policy to allow a service to publish access control requirements as policy statements. While WSDL describes a service and WS-Policy specifies authorization and confidentiality requirements, none of them address access control information.

A policy in this work is a collection of assertions that must be fulfilled to access a service. A service is an object whose methods can be invoked. An assertion describes some aspect about a service user, such as the employee id, the employee's seniority, an associated credit card number, and so on. Assertions are provided by a requestor who is a proxy between a user and the service provider, and who maintains a trust relationship with the provider.

The provided policy language supports the following access control rules

```
cando(Object, Role, SignA)
trust(Requestor, K_R)
request(requestor(Requestor, K_R), assert(attr1, attr2, attr3))
active(Requestor, Role):- trust(Requestor, K_R), request(requestor(
   Requestor, KR), assert(attr1, attr2, attr3))
access(Object):- active(Requestor, Role), cando(Object, Role,
   SignA)
```

Service consumers and providers communicate through SOAP messages augmented with access control information as part of message headers (to decouple application and access control logic). Headers are inspected before each service invocation to perform an authorization check by a logic-based authorization manager (developed as a Prolog inference engine). This manager responds to access requests (e.g., `access(PlaceOrder)?`) by checking the assertions about the user against an internal policy base and accordingly grants or denies access to the web service. The policy base defines access control rules involving object,

methods, subjects, and signed actions (e.g., `cando(ExpediteOrder, management, +exe)`). However, and unlike our approach, Coetzee et al. only regard assertions about the user, but do not address more fine-grained access control based on the properties of arguments that object methods are being invoked with. For example, although physicians may have access to a prescription service, the physician's may only be allowed to prescribe to her own patients, where the patient's name or UUID is the argument to the method *prescribe*.

This approach, like ours, supports formal policy specifications and role-based access control. Each role is defined by a set of assertions; users are matched to roles by matching assertions in the incoming message with assertions about roles. If there is a match between user assertions and policy rules, a specific role is activated, permitting the user to invoke the requested service method. Additionally, policies can be modified and extended without affecting the functionality of the service.

Trust relationships are established between consumer and provider through and initial presentation of credentials. This approach supports changing trust relationships by decoupling trust statements from facts. In our approach, trust can be withdrawn by revoking bestowed CURLs. The advantage that this access control model has over ours is that access control decisions are made at runtime, supporting the dynamic policy base where facts are added and updated. Our approach requires re-issuing user CURLs to adapt to new facts and policies, which depending on how often policies and role membership change, it may be an overhead.

The substantial differences with our approach are that access control information is embedded in message headers, access decisions are based on user assertions, and service requests are mediated by a trusted intermediary. The relative disadvantage compared to our work is that every policy needs to be repeatedly checked every time there is a service request for authorization information. In our approach, policies related to the user's role are checked once while the user's role holds, and only those which depend on the context and the environment are checked at every request; a CURL issued to a user entitles him/her to use a service until

the CURL's expiration or use count exhaustion without having to redundantly check some access control processes. Another drawback is that in this model the service user is required to understand the semantics of the access control requirements; in COASTmed the service consumer is only required to understand the semantics of the service.

This work does not explicitly consider service customization or personalization, only selective access according to user roles.

Coetzee et al. Web Service access control [76]
Problem addressed Address security concerns in web services environments where unknown users request access to services. While WS-Policy defines requirements of authentication and confidentiality, it does not include access control conditions.
Goal Provide access control solutions to protect an organization's resources from unknown users.
Contribution A logic-based access control scheme for web services.
Scope Service providers and consumers.
Architecture type Service oriented
Computing paradigm Object oriented
Used technologies SOAP messages to invoke services (with access control information embedded in message headers); an authorization manager built in Prolog; services are objects whose methods are invoked.
Domain Any

Table 8.9: Coetzee et al. Web Service access control summary

OASIS

OASIS is a role-based access control model which extends traditional RBAC models by supporting decentralized role management, parameterized roles, and role appointment [28][101]. In decentralized RBAC, each organization part of a larger network, manages their own roles locally and roles are service-specific. Decentralization is supported by an event-based architecture and distributed resources are wrapped by OASIS services. Changes in the envi-

ronment are notified to every service through asynchronous messages so that access control policies are checked for compliance at all times.

A significant difference with our approach is that while in COAST-based services policy specification is the liaison between user roles and services, in OASIS services define the roles that are allowed to make use of their methods through defined policies. Users present credentials which are checked against policy specifications. Credentials which satisfy policy constraints lead to role activation (a role membership certificate is issued) and service use authorization. Rules on role membership determine the conditions under which a role is activated for a principal and the conditions that must remain true to avoid role deactivation.

Conditions on role activation may include presenting appointment certificates—long lived credentials bestowing some kind of qualifications—obtained once users have been authenticated. Roles may be appointed through appointment certificates by other authorized roles (e.g., the administration releases an appointment certificate to a hospital physician of the form `employed_doctor(id))`. The difference with more traditional role delegation is that the appointer does not need to hold the role privileges of the appointee (e.g, the administrator does not need to have medical qualifications to appoint a physician).

One of the differences with our capability-based approach is that in OASIS roles are activated in the context of sessions. In COAST, CURLs are issued to principals in particular roles and contain all the information needed to serve a request, thus there is no concept of sessions.

Parameterized roles permit having a fine-grained specification of roles since diverse role parameters or parameter combinations can bestow different privileges or express exceptions to default role privileges (e.g., Dr. Jones may not access my EHR). Parameters may be, for example, id or location of a computer, name and id of a role activator, group membership, pre-requisite roles, patient the activator is treating in the case of a healthcare scenario, and so on. Also, environmental conditions such as the time of day or a temporary status may

expand or limit role activation and consequent service access (e.g., an employe currently on duty: `on_duty(id)`).

Similarly to our techniques, policy rules are formally expressed in a model based on propositional logic. For example, a policy that states that a principal may acquire the role `doctor_on_duty` when she is at work is expressed: `local_user(id?), employed_doctor(id?), on_duty(id)` $\vdash$ `doctor_on_duty(id)`.

OASIS was put into practice within an national health service system where electronic health record fragments are retrieved from within a network of decentralized healthcare providers to obtain integrated patient information. An indexing service maintains a directory of health record fragments hosted at various OASIS-aware healthcare service providers. Privacy is further preserved by anonymizing service requests before retrieving data fragments so that individual providers cannot know the nature of the data requesting organization (e.g, requester may be a HIV specialist). Comparatively, our approach has no difficulty in retrieving information fragments from different sources, and may be in some circumstances more efficient than OASIS since policy checking does not to be done every time, unless policies are request-dependent (as opposed to user-dependent). In OASIS, access control is performed for each data fragment locally. While OASIS maintains a role-based model, COAST is based on capability-based security.

It can be argued that OASIS, to some extent, provides personalization through parameterized roles by defining fine-grained policies according to role parameters. However, OASIS focus is neither personalization nor customization. It does not provide customization capabilities to allow users to tailor the service to their own needs.

OASIS [28][101]
Problem addressed Traditional RBAC models lack decentralized role management, parameterized roles, and role appointment (as opposed to delegation).
Goal Supporting decentralized and secure information access.
Contribution A role-based access control model which supports decentralized role management, parameterized roles, and role appointment.
Scope A role-based access control model for decentralized role management.
Architecture type Event-based
Used technologies For the NHS EHR prototype: XML representation of policies; PostgreSQL Object-Relational database to store active predicates; SOAP messages over HHTPS for asynchronous communication; J2EE for backend.
Domain Any / example in EHR domain

Table 8.10: OASIS summary

PERMIS PMI

PERMIS PMI is a policy-driven, role-based privilege management infrastructure [64]. Policies define recognized system roles, the privileges assigned to each role, the conditions over those privileges, and the parties authorized to assign roles. Policies may include environmental parameters such as date or time of day. Policies are stored in digitally signed attribute certificates (signed by a locally trusted authority) within LDAP directories.

To use services, users present roles (also stored in attribute certificates)—digitally signed by a role-assigning authority—and the PERMIS decision engine checks them against the policies in LDAP directories to accordingly grant or deny access to services. Roles issued by an organization are also hierarchically stored in LDAP directories using distinguished names. Alike COASTmed, PERMIS supports rights inheritance where superior roles have the same authority as roles lower in the role hierarchy. In a service pull mode, as well supported in PERMIS, the decision engine retrieves user roles from pre-configured LDAP directories. COAST-based services can also support pull and push modes—users can send computations to be evaluated by a service, or a long-running or user-spawned service can send messages

to the user based on a user's subscription to the service's notifications.

Policy specifications are semantically based on the Ponder policy language and syntactically specified in a DTD schema. The PERMIS policy elements are: (a) a SubjectPolicy which specifies user subject domains (e.g., National Healthcare professional); (b) a RoleHierarchyPolicy which specifies the recognized policy roles and their hierarchical structure; (c) SOAPolicy which enumerates the trusted authorities to designate roles to subjects (e.g., the National Department of Health); (d) RoleAssignmentPolicy which assigns roles to subjects; (e) TargetPolicy specifies the target domains (e.g., PrescribingApplications); (f) the ActionPolicy specifies the parameterized actions supported by target domains (e.g, Prescribe); (g) TargetAccessPolicy associates roles with authorized actions, corresponding targets, and access conditions. Access control policies explicitly specify methods and arguments supported by each target application (i.e. object). Alike our approach, in PERMIS, everything is denied unless explicitly granted.

The authors claim PERMIS's suitability for large-scale applications given that digitally-signed role assignments can be distributed among participating parties who require them for access control purposes. This model has been used in the experimental context of electronic transmission of drug prescriptions (ETP), where parties involved have different privileges. For example, a pharmacist is only entitled to see the prescriptions that she is responsible to dispense. Roles are assigned to different healthcare providers and government agencies (e.g., physician, decease control agency) and are used by ETP decision engines to grant permissions. Some higher authority (the Secretary of State for Health in the provided example) defines access control policies. The ETP prototype provides services to allow doctors to prescribe drugs (*prescribe* method), for pharmacists to dispense them (*dispense* method), for patients to receive free prescriptions (*dontCharge* method), and perform administration functions (*administrate* method). At incoming service requests, the ETP fetches both the policy and the user's role from configured LDAP directories to make an access control decision.

One of the fundamental differences with our approach is that policies in COASTmed are local to individual healthcare providers, while PERMIS PMI considers situations where there is a higher authority such as the National Department of Health which defines the policy for all participants, and local ETP systems which access a central LDAP directory to fetch this global policy and use it to perform local access control. The drawback is that while there are national disclosure guidelines such as HIPAA, individual institutions have their own privacy policies, thus PERMIS, in this aspect, does not support decentralization. Also, we consider access control for local participants, therefore supporting the division of labor within an organization and hindering insiders' threats. In addition, PERMIS services are rather coarse-grained, one-size-fits-all, and non-customizable as proposed in our model. Finally, a centralized privilege management infrastructure is difficult to manage when there hundreds of roles, targets, and actions in a domain and PERMIS policies may quickly become verbose and complex.

PERMIS PMI [64]
Problem addressed Contextual to the sample domain, Electronic Transmission of Prescriptions do not enforce authorization but trust medical professionals not to abuse their privileges.
Goal Ensure that only medical professionals involved in the Electronic Transmission of Prescriptions can access the system to prescribe drugs to patients.
Contribution A policy-driven, role-based privilege management infrastructure.
Scope Service users and policy making authorities.
Architecture type SOA
Used technologies XML formatted policies; digitally signed X.509 attribute certificate; LDAP directories
Domain Example in electronic transfer of prescriptions

Table 8.11: PERMIS PMI summary

Privacy and Access Control for IHE-Based Systems (Katt et al.)

The Integrating the Healthcare Enterprise (IHE) organization promotes sharing of healthcare information among information systems. This work proposes a security architecture for access control and privacy which supports IHE ontologies, interfaces, and communication standards (e.g., HL7 and DICOM) [155]. This architecture supports and differentiates between standard access control policies and privacy policies. Access control policies support role-based access to resources—mappings between roles and health document types are encoded in a *Rule-Base*. Privacy policies represent patients preferences with respect to the disclosure of their information. For example, a patient may choose to disclose only part of his health record to a doctor or to another party for a specific purpose. COASTmed also supports role-based access to resources, and it can be extended to include patient-specific policies that are taken into account when disclosing their personal information.

In Katt et al., a patient's EHR is a collection of documents or *resource objects* potentially distributed among various healthcare providers or *domains*. Although the EHR pieces are physically distributed, a user within a *requesting domain* perceives an EHR as a single, unified virtual resource. In contrast, information providers within *responding domains* perceive each portion of an EHR as an individual document. Users do not need to check their rights with respect to each distributed resource; a *decision request* is sent to a central *Policy Decision Point*, which fetches individual patient's privacy policy from a *Policy Administration Point*. If the user is entitled to access the entire virtual EHR, a *permit* decision message is sent back to the user. The user sends the request along with this decision to every responding domain so that each EHR portion is fetched to the user. If the user has no access right to every portion of an EHR, the central Policy Decision Point forwards the request to an *Policy Enforcement Point* at each responding domain, which in turn send a *multi-resource* request back to the central Decision Point; a decision on this multi-resource request is done based on the fetched policies. A decision response is the sent in response to the decision request

to each responding domain so that specific EHR portions are returned to the requester.

Policies are described in XACML, one of our evaluated languages (see section 5.2.2). For example, a policy which grants a user full access to a patient's data is described as:

```
<subject category=access-subject> userx_id </subject>
<subejct category=owner-subject> patientx_id <subject>
<resouce> any </resource>
<actoin> read/write </action>
```

Identity assertions as well as Policy Decision Point's decision assertions sent to the Policy Enforcement Points at responding domains are specified in the SAML authorization standard. The architecture components exhibited in a provided prototype are the Policy Enforcement Points within each domain, a central Policy Decision Point, and a central Policy Administration Point which hosts access control and privacy policies.

The advantage of this approach is the ability to make fine-grained access control decisions based on individual EHR documents. However, the described data request process is rather complex and requires numerous messages between enforcement, decision, and administration points, and heavily relies on a centralized Policy Decision Point (single point of failure). In a COAST-based model, CURLs are issued to users by individual "response domains" based on a policy-based access control process. From then on, users use these CURLs to retrieve information from different sources, and locally or remotely integrate it. There is no centralized authority or policy repository nor numerous request messages to obtain the authorized information. In addition, it is not clear how, in Katt et al., information is returned to the user and on whether the user can customize the information services to obtain the required unprocessed or processed data.

Another difference between this work and ours, is that while our focus is on how individual providers can offer services enforcing their own policies, Katt et al. are more concerned with

the aggregation of distributed, decentralized data. However, we do address data integration from sources under different spans of authority as part of our research goals (goal 6 3).

Specifying policies is not a simple endeavor; alike specifying policies in our prototype, it requires domain expertise. However, an as mentioned by the authors, usability studies and easy-to-use interfaces are required to make policy specification feasible in day-to-day practice.

Privacy and Access Control for IHE-Based Systems [155]
Problem addressed It is challenging to provide effective access control mechanisms and support privacy of information in distributed EHR systems.
Goal Supporting a single decision information request with respect to multiple distributed resources composing an EHR.
Contribution A security architecture for access control and in support of patient privacy for distributed IHE systems.
Scope Requesting and responding domains
Architecture type SOA
Prototype A security system which retrieves virtual EHRs from multiple sources.
Used technologies IHE XDS profile for interoperability when sharing EHR; XACML policy language; SAML documents for identity and decision statement assertions.
Domain Electronic Health Records

Table 8.12: Privacy and Access Control for IHE-Based Systems summary

8.2.3 Analysis Results

An overview and evaluation of a set of technologies addressing access control, security, and policies specifications for web services and systems (section 8.2.1) demonstrate that none of these technologies alone can achieve, simultaneously, the goals of differential access and customization of services. These technologies are either abstract models (such as well-known access control models) or standalone languages specifications (e.g., WS-Security and WS-Policy) which are meant to be used in conjunction with other technologies and within more comprehensive architectures, and thus do not provide a holistic solution to our problem. For

example, both WS-Security and WS-Policy extend the semantics of a SOAP messages with security features and requirements, however they do not provide guidance on how services effectively enforce these policies and security requirements.

Most of the analyzed systems (section 8.2.2) have a policy-based approach to the authorized disclosure of patient data—the evaluation of explicitly specified policies in a formal language is the foundation for authorization and access control decisions to access data and invoke restricted system functions. Various formalisms, diverse in semantics and syntax, are used to capture privacy policies. For example, ALDSP supports XQuery and WebLogic security policies; OASIS policy model is based on propositional logic; InfoShare leverages an XPath-like notation; Bhatti et al. choose the X-GTRBAC access control policy specification language, and; Coetzee et al. extend WS-Policy to enable access control for web services. Systems are also different in the expressiveness and the difficulty to specify policies. For example, InfoShare supports the specification of both authorization and prohibition policies, requiring policy conflict resolution, while Cassandra, PERMIS, and COASTmed minimize complexity by allowing only positive policies (i.e. everything is denied unless explicitly granted).

Systems also differ on who is responsible for defining policies. For example, in Haas et al. the patient is exclusively responsible for defining disclosure policies while the data holder neither defines policies nor has access to data. In contrast, in InfoShare and the Privacy and Access Control for IHE-Based Systems (hereafter, Katt et al.) both the data holder (e.g., a hospital) and the patient may specify data disclosure policies. Similarly, Yee et al.'s PPCS allows both web service consumers and providers to define policies, the former regarding the circumstances in which his/her information should be shared and the later on the disclosure policies the user needs to accept in order to use the service; these policies may be in conflict or may be orthogonal. In COASTmed and ALDSP, an organizational policy maker or data service administrator defines the privacy and security policies (although in the future we aim to combine policies defined by both data holders' and individuals' whom the data describes).

Some of the evaluated systems have the ability to provide services that are explicitly tied to and compliant with policies. Systems such as the Policy-Based Authorization Framework (hereafter Bhatti et al.), ALDSP, InfoShare, OASIS, PERMIS PMI, and COASTmed make explicit the association between policies and offered services. For instance, Bhatti et al. defines policy attachments which bind policies and web service components; ALDSP imposes access restrictions on business objects elements and accessor functions, and allows omitting, perturbing, and filtering according to policies the returned information; InfoShare's policies explicitly point to hierarchically structured data objects returned to users; in OASIS, service access is contingent on presenting role membership certificates issued according to policies, where services are associated to roles; in PERMIS, target applications' methods and arguments are specified within access control policies; COASTmed explicitly binds policies with system functions, generates service CURLs based on the evaluated policies, and thus policies are effectively enforced; PPCS ties policies and services by checking for non-conflicting provider and service consumer policies before allowing access to a service.

In contrast, in other systems it is not made explicit how access rights are bound to capabilities. For example, in Cassandra, policy evaluation produces either an "accept" or a "deny" access decision (presumably, access control mechanisms bind policies and services); in Haas et al. patient policies are checked against requests to, accordingly, grant or deny access to data; in Katt et al., a central policy decision makes a "multi-decision accordingly" sent to individual policy enforcement points, but it is not detailed how policy enforcement is carried out and how services are effectively tied to these decisions; alike PPCS, Coetzee et al. allows access to a service according to role assignment based on valid user assertions, but alike all of the above mentioned cases it is not clear how policies and services are explicitly bound.

Most evaluated systems—Bhatti et al., ALDSP, Cassandra, Coetzee et al., PERMIS PMI, and Katt et al.—provide role-specific services addressing user categories or role membership. However, it may also be desirable to treat individuals distinctively despite of their roles (such

as when articulating exceptions), therefore requiring more flexibility than the provided role-based access control. In the Pervasive Healthcare Systems Architecture (hereafter, Rafe et al.), consumers subscribe to services of interest, therefore in some sense services are audience-specific, however not specific to any particular user. A more fine-grained approach to access control considers also subject identity. InfoShare, for example, provides authorized views of patient data to both roles and principals; in OASIS all privileges are associated with both roles and principals; in COASTmed, policy specification is the liaison between user roles and services. However, COASTmed not only supports role based-access control, but policies can be as low level as necessary, specifying per-organization or per-individual policies according to distinctive attributes. In PPCS, the user's privacy policy is user-specific (i.e. the information a user chooses to disclose), however the service itself is the same for all users; what may be user-specific is the provider's disclosure policy thereafter with respect to the user's disclosed information.

With the exception of COASTmed, none of the evaluated systems enable user-controlled customization—services such as in PERMIS are rather coarse-grained, one-size-fits-all. Marmite is a very distinctive system from the other evaluated ones, focusing in service composition rather than on access control. Recall that Marmite is a browser plugging which allows end users to compose mashups. In contrast to COASTmed, Marmite is meant for end users and uses as building blocks traditional, rigid web services. COASTmed is geared towards application programmers building customizable services. In some way Rafe et al. offers customization by allowing users to receive only notifications from events of interest, however they are not personalized nor custom; notifications are either public or private, without further granularity to subscribe to and publish events. Note that while personalization is controlled by the provider, customization gives users some degree of control. OASIS also provides limited personalization through parameterized roles. Distinctively, COASTmed's code mobility enables customization by allowing the user to compose expressive closures which are executed by the service to obtain the required data as described in section 6.2.

	expressively captures policies	policy compliant services	user-specific services	customizable service
TECHNOLOGIES				
SOA orchestration	✗	✗	✗ No authorization semantics in BPEL	✗
SOA choreography	✗	✗	✗ No authorization semantics in WS-CDL	✗
XACML	✓	✓ presumably AC mechanisms between policies and services	✓ authorization based on user's attributes	✗
SAML	✗	✗	✓ may support provision of user-specific services based on identity	✗
WS-Security	✗	✓ if security requirements adhere to WS-Policy specifications (regarding communication security)	✗	✗
WS-Policy	✓	✓ if the service implementation is faithful to advertised policies	✗	✗

RBAC, MAC, ABAC, & DAC	✗	✓ presumably AC mechanisms between AC rules and services	✓ presumably users provided different services according to AC rules	✗
SYSTEMS				
ALDSP	✓	✓ policies enforced by the WLS	✓	✗
X-GTRBAC + WS-Policy	✓	✓ WSDL descriptions bound to policy assertions	✓ role-based AC	✗
Marmite	✗	✗	✓	✓ By allowing to compose mashups from remote web sources.
Cassandra	✓	✓ presumably AC mechanisms between policies and services	✓ role-based AC	✗
Haas et al.	✓ policy specification not described	✓ patient policies checked against request and access to data granted or denied accordingly	✓ patient-specific services and data requester-specific data provided	✗
InfoShare	✓	✓ policies explicitly point to hierarchically structured data objects returned to users	✓ provides authorized views of patient data to requesters (both roles and principals)	✗

Privacy Policy Compliance System (PPCS)	✓ policy specification not described	✓ presumably mediated by a Privacy Controller and a Private Data Import/Export component	✓ very coarsely-grained - data owners and other parties; presumably differential service according to patient policies on disclosure	✗
Coetzee et al. Web Service access control	✓	✓ service name specified in AC policies	✓ role-based AC	✗
OASIS	✓	✓ service access based on role membership certificate presentation; certificate issued according to policies (indirect relationship between policies and services)	✓ all privileges associated with roles and principals	✗
PERMIS PMI	✓	✓ target applications (objects) methods and arguments specified in AC policies	✓ role-based AC	✗

Privacy and Access Control for IHE-Based Systems	✓	✓ presumably mediated by a Policy Decision Point and a Policy Enforcement Point.	✓ role-based AC	✗
Pervasive Healthcare Systems Architecture	✗	N/K	✓ consumers subscribe to services of interest	✓ Partial customization through event subscription

Table 8.13: Evaluation summary

8.2.4 Discussion

Decentralization is at the heart of the problem we seek to mitigate, thus the common trait of the evaluated systems is their focus on decentralization. However, many existing EHR solutions exhibit centralized Supervisory Control and Data Acquisition (SCADA) types of architectures [86]. SCADA supports architectures of distributed sensors and actuators—therefore mostly focused on hardware control—which are governed by a central controller. These architectures are more appropriate when rigorous control is desired, therefore its appropriateness for controlling actuators and acquiring data from distributed sensors in manufacturing plants. These networks are awash in security weaknesses, have a single point of failure, and are subject to frequent attacks [140]. Moreover, they are not meant to capture processes involving complex communities of autonomous parties. They could, however, be components within larger decentralized systems such as COASTmed, for example, for controlling medical devices through policy-based services.

One of the distinctive properties of COASTmed and more generally of COAST-based services is that trust mechanisms are meant to be decentralized. COAST relies on self-certified

islands along with decentralized Web of Trust (WoT) mechanisms, while approaches such as Cassandra rely on centralized certificate authorities. Also, in COASTmed, policies are local to individual healthcare providers, while in PERMIS, for example, a centralized authority such as the National Department of Health defines policies by which local services operate. Another example is Katt et al. which relies on a centralized policy repository to make access control decisions to obtain distributed data.

The analyzed technologies and systems exhibit substantial differences in goals—Katt et al. focuses on the aggregation of distributed data, while COASTmed emphasizes on how individual providers can offer services enforcing their own policies. However, they are all concerned with the authorized and secure access to data. An important insight provided by this evaluation is that technologies focus on either access control (some through formal policies) or on customization, whereas none of them have the dual goal of offering a secure, yet customizable service access. Access control technologies, which constitute most of the analyzed systems and technologies, are focused on allowing service providers to securely expose their resources, however they disregard users' needs. Services are largely unilaterally controlled. In the analyzed literature, access control and service customization are mostly unrelated and detached system concerns. Our intention is to bridge this gap by providing services that benefit and address concerns of both service consumers and providers. COAST-based services stands out by simultaneously enabling differential access to services and user-controlled customization.

One of the limitations that we have found with regards to access control with many of these systems and technologies is that access control can be compared to a gate keeper; once a "permit" decision has been made, access control mechanisms are not specially concerned with how thereafter rules and policies are effectively enforced and maintained. The severity of this issue is that if for some reason access control fails (e.g., human mistake or system bug), the user could gain access to much more information and capability than what was intended.

In systems such as ALDSP, information and capabilities are open to any user unless specific restrictions are tacked on individual elements and functions. In contrast, our COAST-based approach assumes a closed world where everything is prohibited unless explicitly authorized since we assume every piece of information and capability is potentially sensitive. We consider this a more secure approach and less prone to error. COAST-based services are created based on policy evaluation, therefore capabilities that are not authorized for a given user do not even exist in the service's lexical scope. Even if, by mistake, a policy maker provides an unintended right to some party, only the capability associated to that mistaken right will be added to the user's service, therefore a potential security and privacy threat is somewhat mitigated. Of course, the potential consequences of any such unintended right will depend on the nature of the bestowed capability. It is thus important to address access control with respect to web services, mostly of those inter-organizational. Unfortunately, as Bhatti et al. argue, "access control in Web services is a neglected frontier that has not seen the development and adoption of many standards" [47]. It is also argued that although there is a vast body of literature in the area of access control, traditional access control techniques are not appropriate for web services given that web services are inherently different from objects [43] and where users are not known upfront [20].

Another difference between these systems and COASTmed is that access control is a process performed in its entirety at every service request. For example, in Katt et al., a single request requires complex and repetitive policy checking and multiple inter-host messages to access data within distributed providers. In COASTmed, role- and identity-related policies are checked once while the user maintains the same role to obtain a user-specific service CURL. Only time- or date-related policies or those which depend on the properties of the specific service request require evaluation at every request. Additionally, a provider may arbitrarily deny service to a given principal for any reason just by revoking the principal-specific CURL.

However, given that roles within organizations change, it is important to reflect this change

within service CURLs. One approach to this is to revoke CURLs of users whose roles have changed and automatically issue a new CURL with the updated authorizations according to the newly assigned roles (or divest from any right if the user has no assigned roles). A second approach which would make more sense when roles within a domain are highly dynamic, is to simply check all the relevant policies at service use time. Choosing between these approaches is domain dependent, and the choice depends on which approach requires less overhead (check policies versus revoke and reissue CURLs). We have not addressed this topic in depth since we consider the problem of role validity appertaining to service adaptation and evolution which we will pursue in future work.

The evaluated systems have, however, many advantages and features worth adopting in future work. For example, ALDSP not only considers policies to access information and functions, but also post-processes information retrieved by a read operation by filtering unauthorized data; Cassandra considers the exchange and combination of inter-organizational policies; Haas et al.'s approach tags data with provenance information to support the continuity of policy compliance throughout the flow of patient data; InfoShare aggregates health data from different sources, combining their individual policies and resolving conflicting ones in order to comply with all providers' privacy requirements; Katt et al. has the ability to make fine-grained access control decisions based on individual EHR documents whose fragments are dispersed among autonomous providers.

We believe that one of the barriers for adoption of some technologies, including COASTmed, is the complexity of formalizing policies. However, the critical nature of the information justifies this difficulty. Policy languages have undeniable advantages such as unambiguously capturing domain rules and automating policy evaluation and conflict resolution. Health care systems are critical and increasingly ubiquitous; ensuring their reliability, quality, accuracy is crucial [219]. Formal policies, specified by expert teams of domain policy makers and developers, are essential to ensure these properties, thus guiding the correct system behavior.

To facilitate their specification, appropriate user interfaces and policy reuse are needed to support policy makers in their tasks as well as basic user interfaces to capture individual's policies (such as patient consent) to make policy specification feasible in day-to-day practice.

8.3 Generalizability to Other Domains

Our evaluations, in specific the practical experiments with COASTmed and the scenario-based evaluations, are strong indicators that our proposed model is applicable to real systems and generalizable to other equally dynamic domains where the salient difference is the type of data exchanged. Our approach to the safe and customizable access to personal information can be extended to other domains which experience similar challenges (see section 2), such as the financial and governmental worlds. For example, consider Kukafka et al.'s scenario where healthcare organizations gather real-time data for disease surveillance [162]. In another hypotetical scenario, government agencies such as the Bureau of Economic Analysis or the Federal Reserve are interested in monitoring individuals' saving ability by comparing disposable income vs. savings across time. These agencies may send custom computations to banks and credit unions that, without revealing the identity of customers, provide them with this information.

We focus on individuals' information because it clearly depicts the challenges related to access and sharing of private data. However, this problem can be related to any type of data. For example, government agencies such as DARPA, the State Department, the CIA, the FBI, and so on, hold classified documentation whose access depends on rigorous authorizations according to various levels of security clearance. Another example is e-commerce, where customers have various levels of service access based on the particular terms of their subscriptions. The National Institute for Standards and Technology (NIST) for instance produced a report which exhibits the information access control needs of a wide array of

public and private organizations in domains such as "energy, financial, communications, insurance, manufacturing, computers, and services" for "law enforcement, benefits delivery, medical/hospital, nuclear/energy, space exploration, defense, tax system, information collection and dissemination, air traffic, and service center operations" activities [104]. Differential access and service customization capabilities can thus be leveraged and applied to any domain with complex relationships between individuals, organizations, and information where the key guiding principles for party interaction are trust and authority.

Chapter 9

Conclusion

In these studies, we seek to solve problems regarding the safe disclosure and meaningful use of personal information, which concern both consumers and providers of personal data services. First, trust between consumers and providers is complicated, nuanced, and not homogenous, thus the management of an organization's data disclosures is highly complex and requires complying with ethical and legal policies. Although an organization's data disclosure policies may be somewhere described, they are often isolated from information systems' operations, and therefore services for users within and foreign to an organization do not always comply with these regulations. Compounding the problem is the difficulty to capture these nuanced trust relationships with current technologies and provide per-user services that are bound to organizational privacy policies. Second, current information services are rigid, "one-size-fits-all" solutions that do not meet the ever-changing needs of a diverse pool of people and organizations, whereas information is used for myriad, divergent, and unforeseen purposes. This is the result of providers lacking the ability to differentiate among users.

In light of these challenges, our work seeks to mediate this tension between information need and service provision through dynamic architectures that enable customization, while

simultaneously provide service access based on specific trust relationships. Our ultimate goal is to provide flexible and customizable, yet secure and privacy-aware Internet-based data services which operate according to organizational policies. The system properties that we pursue are therefore twofold: (a) **policy-based differential access**, which enables providers to treat users individually by providing privacy-aware services that make available authorized information and computation capability while deny the access to forbidden assets according to formally defined policies, and; (b) **service customizability**, which allows individual users to computationally manipulate personal information to fulfill specific data needs, within the authority granted by the provider according to individual rights to access data according to ownership, trust, and legal relations.

Our solution to these challenges consist on an architecture-centric approach, where a combination of novel Software Architecture principles and policy languages enable building services which elicit the desired, above-described properties. Our work on secure and customizable Internet-based services is rooted on the principles of the COAST architectural style [121]—which succeeds current Web architectures to meet increasing openness, flexibility, dynamicity, and security demands—along with the expressiveness of the Rei policy language to formalize organizational policies describing data disclosure and service access conditions.

Guiding these studies is the characterization of the principal design decisions for engineering services which simultaneously enable customization in order to fulfill diverse information needs and policy-compliant differential service access. The elements of our model which, together, approach the challenges set forth are:

1. expressively **capturing privacy and operational policies** using an appropriate formal policy language such as Rei (sections 6.1.1 and 7.3); our modified version of Rei allows associating subjects (identities or roles, e.g., physician), actions (e.g., prescribe), direct and indirect action targets (e.g., NSAID to patient), and a set of binding con-

ditions (e.g., if physician is the patient's primary care physician); we adopt a "closed-world" assumption, where everything is forbidden unless explicitly authorized (thus we ignore Rei's prohibitive policies);

2. **associating** a system's **functional capabilities** with a set of provider-defined privacy and operational **policies** with the goal of controlling the access to these capabilities (section 6.1.5); capabilities (e.g., `EHR/new`) and policy rights (e.g., the right to create a new EHR) can be explicitly or implicitly associated—where explicit associations make a direct relation between a capability identifier within a database table (e.g., capability `EHR/create` and right EHR-create), an implicit association has no need to lookup database tables since capability and policy right are linked by homonymity.

3. leveraging COAST's Capability URLs and formal policies to **create tokens which allow accessing user-specific services** (sections 6.1.2, 6.1.4, and 7.4); CURLs embed authority information—i.e. the capabilities a principal is entitled to use—that enables differentiating among service users or user types and specializing service offerings; these user-specific CURLs are dynamically generated based on the relevant policies for a given user and the aforementioned association of policies and capabilities; policies are evaluated to obtain the truth values of policy conditions according to a particular user and context, and the capability information corresponding to the associated policies which evaluate to *true* are embedded within issued CURLs.

4. exploiting COAST's binding environment sculpting to **expose functional capabilities as services** (section 6.1.5); a binding environment provides the lexical scope for a user-specific service, namely a set of key-value pairs corresponding to the reachable functions and data within a service; sculpting in this context is composing a binding environment from a chosen subset of the universe of system capabilities; by virtue of environment sculpting, it is not possible for a user to access unauthorized capabilities, since these forbidden bindings are simply absent from a bounded user service;

5. leveraging COAST's capability-based security—CURLs and binding environments—to **dynamically create bounded, user-specific services** based on the authorization semantics embedded in CURLs (sections 6.1.6 and 7.5); context specific policies are evaluated at service use to accordingly allow or deny access to a service; if all contextual policies evaluate to *true*, the capability information within CURLs is used to spin up a new service (computation) which has that set of capabilities as a binding environment; following, the user's message is forwarded to these newly created service for execution; given that CURL creation and service use are contingent on policy evaluation and that unauthorized capabilities do not exist within user services, policies are enforced and directly bound to services;

6. exploiting computation composition and mobility to **allow users to create custom services** and obtain the required data through user-created closures built on the authorized capabilities (section 6.2); customization was naturally enabled by COAST—given that the style permits the exchange of mobile code between decentralized peers and that services are computational environments (i.e. can execute mobile code sent in messages), users can exploit functional composition by using the available capabilities made available (in binding environments) by the service provider as building blocks to create more complex functions to process and manipulate the accessible data.

The contributions and outcomes of this research work are, therefore,

- novel techniques for binding formally defined privacy and operational policies to organizations' Web-based personal data services (section 6.1);

- an architecturally sound approach which, by way of architectural principles and a formal policy language, simultaneously enables provider-controlled policy-based differential access to personal data services and user-controlled service customization;

this architectures enables the secure, privacy-aware, customizable use and sharing of personal information in decentralized, trust-dependent domains (section 6);

- design guidance for using the developed techniques in practice by way of the COASTmed prototype; this healthcare application allows formally specifying policies that are bound to specific capabilities and permits users to achieve the desired customization given their individual authority to access these services (section 7);

- insights on the advantages, limitations, and application contexts of our model compared to other systems and technologies with similar goals (sections 8.1 and 8.2).

We tested, through a set of evaluation methods, our hypothesis that a software system rooted in dynamic architectural principles involving capability-based security and computation exchange, along with formal policy specifications can offer policy-based differential access to personal information services, while simultaneously enabling user-controlled service customization in decentralized contexts. These methods included:

1. the experimental assessment of the practical feasibility of the proposed techniques for achieving policy-based differential service provision and fine-grained customization through prototype development; successfully, we developed COASTmed, a notional EHR management system which elicits the desired properties—provides customizable services to diverse type of users with distinct access privileges (section 7);

2. scenario-based evaluations which leverage specific bounded scenarios or vignettes in the healthcare domain regarding complex inter-agency interactions and access control (section 8.1). We captured the policies guiding these interactions and simulated these scenarios using COASTmed as the evaluation artifact. The expected system behavior was observed in every scenario—parties could successfully access and use the authorized information, while unauthorized capabilities were effectively denied;

3. qualitative comparative analyses with respect to the desired properties—differential service access and user-controlled service customization—with systems approaching similar challenges (section 8.2). The most important result in this portion of the evaluation is that we have not found, to this date, a comparable system which provides user-specific, privacy-aware services that simultaneously allow users to customize their experience with the service through rich computational manipulation.

The main question we embarked to answer to is: does our approach enable the different participating parties in a given domain to access the information and computation capabilities they need, in the way and at the time required, and within the boundaries imposed by the service provider by law or for the sake of privacy? The answer to this question is yes; our approach based on principles of computational exchange and capability-based security along with formal policies elicit, in systems offering Internet-based services, the pursued system properties: services which treat users individually and provide them with the ability of tailoring services to their need within some imposed boundaries. Also, we have provided answers to more detailed research questions set forth in section 3:

1. information services providers can have fine grained control over the disclosure of their data and the access capabilities by: (a) formally capturing disclosure policies that are based on role membership or other environment and action's target conditions (section 7.3); (b) associating policies and capabilities to provide privacy-aware services that differentiate users (sections 6.1, 7.4, and 7.5), and; (c) allowing the provider to revoke privileges through provided capability accounting mechanisms (sections 6.1.7,7.4.2, and 7.5.5). In these services, trespassing authorization rights is simply not possible.

2. safe code mobility and functional composition—supported by the COAST infrastructure—are the enabling technologies for using, customizing, and integrating capabilities of one or more decentralized computations (section 6.2).

3. the development of the COASTmed prototype is the evidence that a system that provides services exhibiting differential access and customizability can be built in practice (section 7). The Motile/Island infrastructure provided by COAST and the Rei policy language are appropriate technologies for obtaining these desired properties and support multiple parties, multiple uses of personal information, and different trust relationships. However, these are suggested technologies and do not exclude others that may as well maintain the properties of the COAST style. For example, other domains may find another policy language more suited to a particular type of policies or other frameworks supporting mobile code and asynchronous communications may also be appropriate for building style-compliant services.

The context of this work is decentralized information systems, whose constituent services operate under multiple, distinct authorities. This work is relevant within Service Oriented Architectures, yet our approach is fundamentally different from current uniform, rigid, and unilaterally provided Web services. COAST-based services are computationally enabled, customizable, user-specific, and expose personal data and system capabilities according to formal policies. Our work applies access control features—providers define per-user authority over their services—and it is further motivated by the insufficiency of traditional access control models for complex, large-scale decentralized domains such as healthcare [15].

In addition, this work approaches essential requirements of healthcare applications, namely authorization, data confidentiality, availability, utility, and access to the legitimate use of health records [278][15]. However, our approach not only provides enormous opportunities in the area of healthcare—made evident by the number of EHR and related companies arising (e.g., Practice Fusion, Epic, and NextGen)—but within many domains where complex party interactions are guided by a multitude of trust relationships and access strictures.

Our research naturally leads to extensive and interesting future work such as on "break the

glass" policies, policy conflict and resolution, service evolution, information provenance and traceability, patterns of information integration, and privacy-aware systems. Also, we have considered policies that allow accessing services, but not policies that act upon the results of such computations to therefore decide whether the information can be safely returned to the user or if providing this information would violate some privacy policy.

The main take away from our work is that the access and management of personal information in heterogeneous trust domains goes beyond individual web services and access control technologies, but requires more complex solutions based on sound architectural design founded on coherent architectural principles such as the presented model. Design and rationale prevails and guides all software activities, from requirements, to architectural conception, to implementation, and testing, allowing to elicit, maintain, and evolve the desired system properties [252].

Bibliography

[1] DICOM. http://www.dicomlibrary.com/.

[2] European committee for standardization (CEN). http://standards.cen.eu/index.html.

[3] GNUmed. http://wiki.gnumed.de/bin/view/gnumed.

[4] Kissmetrics. https://www.kissmetrics.com/.

[5] LOINC. http://loinc.org/.

[6] OpenEMR project. http://www.open-emr.org/.

[7] OpenMRS. http://openmrs.org/.

[8] Racklog: Prolog-style logic programming. http://docs.racket-lang.org/racklog/.

[9] trak.io. http://trak.io/.

[10] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Conference on Information and Knowledge Management*, Atlanta, Georgia, 2001.

[11] S. Agarwal, S. Lamparter, and R. Studer. Making web services tradable: A policy-based approach for specifying preferences on web service properties. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):11–20, 2009.

[12] C. C. Aggarwal and S. Y. Philip. *A general survey of privacy-preserving data mining models and algorithms*. Springer, 2008.

[13] G. A. Agha. *Actors: a model of concurrent computation in distributed systems*. PhD thesis, 1985.

[14] S. Al-zharani, S. Sarasvady, H. Chandra, and P. Pichappan. Controlled EHR access in secured health information system. In *Digital Information Management, 2006 1st International Conference on*, pages 63–68, 2006.

[15] B. Alhaqbani and C. Fidge. Access control requirements for processing electronic health records. In B. Benatallah and H.-Y. Paik, editors, *Business Process Management Workshops*, number 4928 in Lecture Notes in Computer Science, pages 371–382. Springer Berlin Heidelberg, 2008.

[16] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer Publishing Company, Incorporated, 2010.

[17] A. Anderson and A. Anderson. A comparison of two privacy policy languages: Epal and xacml. In *In SWS 06: Proceedings of the 3rd ACM workshop on Secure web services*, pages 53–60. ACM Press, 2006.

[18] A. H. Anderson. An introduction to the web services policy language (wspl). In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 189–192, 2004.

[19] M. Aoyama, S. Weerawarana, H. Maruyama, C. Szyperski, K. Sullivan, and D. Lea. Web services engineering: promises and challenges. In *Proceedings of the 24th international conference on Software engineering*, pages 647–648, 2002.

[20] C. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, and M. Verdicchio. Expressive and deployable access control in open web service applications. *IEEE Transactions on Services Computing*, 4(2):96–109, Apr. 2011.

[21] E. Arroyo, T. Selker, and W. Wei. Usability tool for analysis of web designs using mouse tracks. In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*, pages 484–489. ACM, 2006.

[22] P. Ashley, S. Hada, G. Karjoth, and C. Powers. The enterprise privacy authorization language (EPAL).

[23] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (EPAL 1.2). *Submission to W3C*, 2003.

[24] Astronaut LLC. AstronautVista. http://astronautvista.com/.

[25] I. Astrova, N. Korda, and A. Kalja. Storing OWL ontologies in SQL relational databases. *International Journal of Electrical, Computer and Systems Engineering*, 1(4):242–247, 2007.

[26] R. Atterer, M. Wnuk, and A. Schmidt. Knowing the user's every move: User activity tracking for website usability evaluation and implicit interaction. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 203–212, New York, NY, USA, 2006. ACM.

[27] M. Backes, G. Karjoth, W. Bagga, and M. Schunter. Efficient comparison of enterprise privacy policies. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 375–382, 2004.

[28] J. Bacon, K. Moody, and W. Yao. Access control and trust in the use of widely distributed services. In R. Guerraoui, editor, *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 295–310. Springer Berlin Heidelberg, 2001.

[29] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, and others. Web services policy 1.2-framework (WS-policy). *W3C Member Submission*, 25:12, 2006.

[30] A. Baquero, A. M. Schiffman, and J. Shrager. Blend me in: Privacy-preserving input generalization for personalized online services. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 51–60. IEEE, 2013.

[31] H. P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Cambridge University Press, June 2013.

[32] A. Barros, M. Dumas, and P. Oaks. A critical overview of the web services choreography description language. *BPTrends Newsletter*, 3:1–24, 2005.

[33] D. Basin, F. Klaedtke, and S. Mller. Monitoring security policies with metric first-order temporal logic. In *Proceedings of the 15th ACM symposium on Access control models and technologies*, pages 23–34, 2010.

[34] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, New York, 2003.

[35] D. W. Bates, M. Ebell, E. Gotlieb, J. Zapp, and H. C. Mullins. A proposal for electronic medical records in US primary care. *Journal of the American Medical Informatics Association*, 10(1):1–10, 2003.

[36] T. Beale. Archetypes: Constraint-based domain models for future-proof information systems. In *OOPSLA 2002 workshop on behavioural semantics*, volume 105, 2002.

[37] M. Becker and P. Sewell. Cassandra: flexible trust management, applied to electronic health records. In *17th IEEE Computer Security Foundations Workshop, 2004. Proceedings*, pages 139–154, 2004.

[38] M. Y. Becker, C. Fournet, and A. D. Gordon. Design and semantics of a decentralized authorization language. In *Computer Security Foundations Symposium, 2007. CSF'07. 20th IEEE*, pages 3–15, 2007.

[39] I. Z. Ben-Shaul and G. E. Kaiser. A paradigm for decentralized process modeling and its realization in the oz environment. In *Proceedings of the 16th international conference on Software engineering*, pages 179–188. IEEE Computer Society Press, 1994.

[40] B. Benatallah, Q. Z. Sheng, and M. Dumas. The self-serv environment for web services composition. *Internet Computing, IEEE*, 7(1):40–48, 2003.

[41] M. Benyoucef, C. Kuziemsky, A. A. Rad, and A. Elsabbahi. Modeling healthcare processes as service orchestrations and choreographies. *Business Process Management Journal*, 17(4):568–597, July 2011.

[42] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, 1994.

[43] E. Bertino and A. C. Squicciarini. A flexible access control model for web services. In H. Christiansen, M.-S. Hacid, T. Andreasen, and H. L. Larsen, editors, *Flexible Query Answering Systems*, number 3055 in Lecture Notes in Computer Science, pages 13–16. Springer Berlin Heidelberg, Jan. 2004.

[44] K. Bhargavan, C. Fournet, and A. D. Gordon. Verified reference implementations of WS-security protocols. In *Web Services and Formal Methods*, pages 88–106. Springer, 2006.

[45] R. Bhatti and T. Grandison. Towards improved privacy policy coverage in healthcare using policy refinement. In *Secure Data Management*, pages 158–173. Springer, 2007.

[46] R. Bhatti, A. Samuel, M. Y. Eltabakh, H. Amjad, and A. Ghafoor. Engineering a policy-based system for federated healthcare databases. *Knowledge and Data Engineering, IEEE Transactions on*, 19(9):1288–1304, 2007.

[47] R. Bhatti, D. Sanz, E. Bertino, and A. Ghafoor. A policy-based authorization framework for web services: Integrating XGTRBAC and WS-policy. In *IEEE International Conference on Web Services, 2007. ICWS 2007*, pages 447–454, July 2007.

[48] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[49] B. Blobel. Authorisation and access control for electronic health record systems. *International Journal of Medical Informatics*, 73(3):251–257, Mar. 2004.

[50] B. Blobel. Architectural approach to eHealth for enabling paradigm changes in health. *Methods of information in medicine*, 49(2):123, 2010.

[51] B. Blobel and P. Pharow. Analysis and evaluation of EHR approaches. *Methods of information in medicine*, 48(2):162, 2009.

[52] D. Blumenthal and M. Tavenner. The meaningful use regulation for electronic health records. *New England Journal of Medicine*, 363(6):501–504, 2010.

[53] A. C. Bomberger, W. S. Frantz, A. C. Hardy, N. Hardy, C. R. Landau, and J. S. Shapiro. The KeyKOS nanokernel architecture. In *USENIX Workshop on Microkernels and Other Kernel Architectures*, pages 95–112, 1992.

[54] P. Bonatti, J. L. De Coi, D. Olmedilla, and L. Sauro. A rule-based trust negotiation system. *Knowledge and Data Engineering, IEEE Transactions on*, 22(11):1507–1520, 2010.

[55] P. A. Bonatti, J. L. De Coi, D. Olmedilla, and L. Sauro. Rule-based policy representations and reasoning. In *Semantic techniques for the web*, pages 201–232. Springer, 2009.

[56] M. Boniface and P. Wilken. ARTEMIS: Towards a secure interoperability infrastructure for healthcare information systems. *Studies in Health Technology and Informatics*, 112:181–189, 2005.

[57] A. Boonstra and M. Broekhuis. Barriers to the acceptance of electronic medical records by physicians from systematic review to taxonomy and interventions. *BMC Health Services Research*, 10(1):231, Aug. 2010.

[58] V. Borkar, M. Carey, D. Engovatov, D. Lychagin, P. Reveliotis, J. Spiegel, S. Thatte, and T. Westmann. Access control in the aqualogic data services platform. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 939–946. ACM, 2009.

[59] A. Caforio, A. Corallo, G. Elia, and G. Solazzo. Service customization supporting an adaptive information system. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 342–349, 2004.

[60] S. Cantor, I. J. Kemp, N. R. Philpott, and E. Maler. Assertions and protocols for the oasis security assertion markup language. *OASIS Standard (March 2005)*, 2005.

[61] J. Cao, J. Wang, K. Law, S. Zhang, and M. Li. An interactive service customization model. *Information and Software Technology*, 48(4):280–296, 2006.

[62] G. Caronni. Walking the web of trust. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000.(WET ICE 2000). Proeedings. IEEE 9th International Workshops on*, pages 153–158, 2000.

[63] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[64] D. Chadwick and D. Mundy. Policy based electronic transmission of prescriptions. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003*, pages 197–206, 2003.

[65] B. Chandrasekaran, J. Josephson, and V. Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems and their Applications*, 14(1):20–26, 1999.

[66] Q. Chen, J. Grundy, and J. Hosking. An e-whiteboard application to support early design-stage sketching of UML diagrams. In *Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on*, pages 219–226. IEEE, 2003.

[67] R. Chen, G. O. Klein, E. Sundvall, D. Karlsson, and H. Ahlfeldt. Archetype-based conversion of EHR content models: pilot experience with a regional EHR system. *BMC medical informatics and decision making*, 9(1):33, 2009.

[68] X. Chen, D. Berry, and W. Grimson. Identity management to support access control in e-health systems. In J. V. Sloten, P. Verdonck, M. Nyssen, and J. Haueisen, editors, *4th European Conference of the International Federation for Medical and Biological Engineering*, number 22 in IFMBE Proceedings, pages 880–886. Springer Berlin Heidelberg, Jan. 2009.

[69] Y. Chen and H. Xu. Privacy management in dynamic groups: Understanding information privacy in medical practices. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 541–552, New York, NY, USA, 2013. ACM.

[70] Y. B. Choi, K. E. Capitan, J. S. Krause, and M. M. Streeper. Challenges associated with privacy in health care industry: implementation of HIPAA and the security rules. *Journal of Medical Systems*, 30(1):57–64, 2006.

[71] S.-C. Chou and J.-Y. Jhu. Access control policy embedded composition algorithm for web services. In *Advanced Information Management and Service (IMS), 2010 6th International Conference on*, pages 54–59. IEEE, 2010.

[72] A. Church. *The calculi of lambda-conversion*. Number 6. Princeton University Press, 1985.

[73] K. J. Cios and G. William Moore. Uniqueness of medical data mining. *Artificial intelligence in medicine*, 26(1):1–24, 2002.

[74] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):28–34, 2002.

[75] CNN Wire Staff. Hospital personnel fired for accessing records of tucson victims. *CNN*, Jan. 2011.

[76] M. Coetzee and J. H. P. Eloff. Towards web service access control. *Computers & Security*, 23(7):559–570, Oct. 2004.

[77] O. Corcho, M. Fernndez-Lpez, and A. Gmez-Prez. Methodologies, tools and languages for building ontologies: Where is their meeting point? *Data Knowl. Eng.*, 46(1):41–64, July 2003.

[78] . Corcho, M. Fernndez-Lpez, A. Gmez-Prez, and . Vicente. WebODE: An integrated workbench for ontology representation, reasoning, and exchange. In A. Gmez-Prez and V. R. Benjamins, editors, *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, number 2473 in Lecture Notes in Computer Science, pages 138–153. Springer Berlin Heidelberg, Jan. 2002.

[79] L. Cranor, M. Langheinrich, and M. Marchiori. A p3p preference exchange language 1.0 (APPEL1. 0). *W3C working draft*, 15, 2002.

[80] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The platform for privacy preferences 1.0 (p3p1.0) specification. Technical report, W3C, 2002.

[81] L. F. Cranor. P3p: Making privacy policies more useful. *Security & Privacy, IEEE*, 1(6):50–55, 2003.

[82] A. Cser. XACML is dead | forrester blogs. http://blogs.forrester.com/andras_cser/13-05-07-xacml_is_dead, May 2013.

[83] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE*, 6(2):86–93, 2002.

[84] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks*, pages 18–38. Springer, 2001.

[85] C. H. Damm, K. M. Hansen, and M. Thomsen. Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 518–525. ACM, 2000.

[86] A. Daneels and W. Salter. What is SCADA. In *International Conference on Accelerator and Large Experimental Physics Control Systems*, pages 339–343, 1999.

[87] E. Dashofy. The myx architectural style. Technical report, University of California, Irvine, 2006.

[88] E. Dashofy, H. Asuncion, S. Hendrickson, G. Suryanarayana, J. Georgas, and R. Taylor. ArchStudio 4: An architecture-based meta-modeling environment. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 67–68. IEEE Computer Society, 2007.

[89] M. A. C. Dekker and S. Etalle. Audit-based access control for electronic health records. *Electronic Notes in Theoretical Computer Science*, 168:221–236, Feb. 2007.

[90] C. M. DesRoches, E. G. Campbell, S. R. Rao, K. Donelan, T. G. Ferris, A. Jha, R. Kaushal, D. E. Levy, S. Rosenbaum, and A. E. Shields. Electronic health records in ambulatory carea national survey of physicians. *New England Journal of Medicine*, 359(1):50–60, 2008.

[91] D. Detmer, M. Bloomrosen, B. Raymond, and P. Tang. Integrated personal health records: Transformative tools for consumer-centric care. *BMC Medical Informatics and Decision Making*, 8(1):45, Oct. 2008.

[92] P. Dewan and P. Dasgupta. P2p reputation management using distributed identities and decentralized recommendation chains. *Knowledge and Data Engineering, IEEE Transactions on*, 22(7):1000–1013, 2010.

[93] C. Dimarco, G. Hirst, L. Wanner, and J. Wilkinson. HealthDoc: Customizing patient information and health education by medical condition and personal characteristics. In *In First International Workshop on Artificial Intelligence in Patient Education*, 1995.

[94] N. Dunlop, J. Indulska, and K. Raymond. Dynamic conflict detection in policy-based management systems. In *in Sixth International Enterprixe Distributed Computing Conference, 2002*, pages 15–26, 2002.

[95] C. Dwork. Differential privacy: A survey of results. *Theory and Applications of Models of Computation*, pages 1–19, 2008.

[96] S. J. Dwyer III, A. C. Weaver, and K. K. Hughes. Health insurance portability and accountability act. *Security Issues in the Digital Medical Enterprise*, 2004.

[97] M. Eichelberg, T. Aden, J. Riesmeier, A. Dogac, and G. B. Laleci. A survey and analysis of electronic healthcare record standards. *ACM Computing Surveys (CSUR)*, 37(4):277–315, 2005.

[98] W. Emmerich. Software engineering and middleware: a roadmap. In *Proceedings of the Conference on The future of Software engineering*, pages 117–129, 2000.

[99] J. R. Erenkrantz, M. Gorlick, G. Suryanarayana, and R. N. Taylor. From representations to computations: the evolution of web architectures. In *Proceedings of the ACM SIGSOFT symposium on The foundations of software engineering*, pages 255–264, Dubrovnik, Croatia, 2007.

[100] T. Erl. Service-oriented architecture: concepts, technology, and design. *Prentice Hall*, 31:W3C, 2005.

[101] D. M. Eyers, J. Bacon, and K. Moody. OASIS role-based access control for electronic health records. *IEE Proceedings-Software*, 153(1):16–23, 2006.

[102] C. Farkas and S. Jajodia. The inference problem: a survey. *ACM SIGKDD Explorations Newsletter*, 4(2):6–11, 2002.

[103] D. Ferraiolo, J. Cugini, and D. R. Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 241–48, 1995.

[104] D. F. Ferraiolo, D. M. Gilbert, and N. Lynch. An examination of federal and commercial access control policy needs. In *NIST-NCSC National Computer Security Conference*, page 107, 1995.

[105] D. F. Ferraiolo and D. R. Kuhn. Role-based access controls. *arXiv preprint arXiv:0903.2171*, 2009.

[106] A. Ferreira, R. Cruz-Correia, L. Antunes, P. Farinha, E. Oliveira-Palhares, D. W. Chadwick, and A. Costa-Pereira. How to break access control in a controlled manner. In *Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems*, CBMS '06, pages 847–854, Washington, DC, USA, 2006. IEEE Computer Society.

[107] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[108] S. E. Fienberg and J. McIntyre. Data swapping: Variations on a theme by dalenius and reiss. In *Privacy in statistical databases*, pages 14–29, 2004.

[109] P. W. Fong. Relationship-based access control: Protection model and policy language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, CODASPY '11, pages 191–202, New York, NY, USA, 2011. ACM.

[110] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005.

[111] FreeMED Software Foundation. FreeMED.

[112] A. Fuggetta, G. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[113] D. M. Gabbay and F. Guenthner. *Handbook of Philosophical Logic*. Springer Science & Business Media, Mar. 2004.

[114] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Reading, MA, 1995.

[115] C. L. Gardner and P. F. Pearce. Customization of electronic medical record templates to improve end-user satisfaction. *Computers Informatics Nursing*, 31(3):115–121, 2013.

[116] D. Garlan. Software architecture: A roadmap. In *International Conference on Software Engineering (ICSE 2000): The Future of Software Engineering Volume*, pages 91–101, Limerick, Ireland, 2000. ACM Press.

[117] D. Garlan and M. Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, pages 1–39. World Scientific Publishing Company, Singapore, 1993.

[118] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *The Semantic Web: Research and Applications*, pages 342–356. Springer, 2004.

[119] M. R. Genesereth and R. E. Fikes. Knowledge interchange format-version 3.0: reference manual, 1992.

[120] Google. Google analytics.

[121] M. M. Gorlick, K. Strasser, and R. N. Taylor. Coast: An architectural style for decentralized on-demand tailored services. In *Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 71–80, 2012.

[122] M. M. Gorlick, K. Strasser, and R. N. Taylor. Motile: Mobile code to support an architectural style for adaptive decentralized applications. Technical report, Institute for Software Research, 2012.

[123] M. M. Gorlick and R. N. Taylor. Communication and capability URLs in COAST-based decentralized services. In C. Pautasso, E. Wilde, and R. Alarcon, editors, *REST: Advanced Research Topics and Practical Applications*, pages 9–25. Springer, 2014.

[124] O. M. Group. Unified modeling language.

[125] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web*, pages 403–412, 2004.

[126] R. H. Guttman, A. G. Moukas, and P. Maes. Agent-mediated electronic commerce: a survey. *The Knowledge Engineering Review*, 13(02):147–159, 1998.

[127] F. Gmez Mrmol and G. Martnez Prez. Towards pre-standardization of trust and reputation models for distributed and heterogeneous systems. *Computer Standards & Interfaces*, 32(4):185–196, 2010.

[128] S. Haas, S. Wohlgemuth, I. Echizen, N. Sonehara, and G. Mller. Aspects of privacy for electronic health records. *International journal of medical informatics*, 80(2):e26–31, Feb. 2011.

[129] M. Hafner, M. Memon, and M. Alam. Modeling and enforcing advanced access control policies in healthcare systems with sectet. In *Models in Software Engineering*, pages 132–144. Springer, 2008.

[130] J. D. Halamka, K. D. Mandl, and P. C. Tang. Early experiences with personal health records. *Journal of the American Medical Informatics Association*, 15(1):1–7, Jan. 2008.

[131] W. E. Hammond. The making and adoption of health data standards. *Health Affairs*, 24(5):1205–1213, Sept. 2005.

[132] K. Hamouid and K. Adi. Self-certified based trust establishment scheme in ad-hoc networks. In *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pages 1–7, 2012.

[133] J. E. Hanson, P. Nandi, and D. W. Levine. Conversation-enabled web services for agents and e-business. In *International Conference on Internet Computing*, pages 791–796, 2002.

[134] R. Hillestad, J. Bigelow, A. Bower, F. Girosi, R. Meili, R. Scoville, and R. Taylor. Can electronic medical record systems transform health care? potential health benefits, savings, and costs. *Health Affairs*, 24(5):1103–1117, 2005.

[135] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and [lambda]-calculus*, volume 1. CUP Archive, 1986.

[136] HL7 Standards. Role-based access control healthcare permission catalog (RBAC).

[137] A. Hoerbst and E. Ammenwerth. Electronic health records. *Methods Inf Med*, 49(4):320–336, 2010.

[138] R. Hull, B. Kumar, D. Lieuwen, P. F. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas. everything personal, not just business: improving user experience through rule-based service customization. In *Service-Oriented Computing-ICSOC 2003*, pages 149–164. Springer, 2003.

[139] G.-H. Hwang, C. Wu-Lee, and Z.-X. Jiang. Workflow-based dynamic access control in a service-oriented architecture. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 47–52, 2012.

[140] V. M. Igure, S. A. Laughter, and R. D. Williams. Security issues in SCADA networks. *Computers & Security*, 25(7):498–506, Oct. 2006.

[141] ISO/TC 215. ISO/TR 20514:2005, health informatics - electronic health record - definition, scope and context. Technical report, International Organization for Standardization, 2005.

[142] T. Jaeger, A. Edwards, and X. Zhang. Managing access control policies using access control spaces. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 3–12. ACM, 2002.

[143] J. Jin, G.-J. Ahn, H. Hu, M. J. Covington, and X. Zhang. Patient-centric authorization framework for sharing electronic health records. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 125–134, New York, NY, USA, 2009. ACM.

[144] M. Johnson, P. Chang, R. Jeffers, J. M. Bradshaw, V. W. Soo, M. R. Breedy, L. Bunch, S. Kulkarni, J. Lott, N. Suri, and others. KAoS semantic policy and domain services: An application of DAML to web services-based grid architectures. In *Proceedings of the AAMAS*, volume 3, 2003.

[145] W. Jones. Personal information management. *Annual review of information science and technology*, 41(1):453–504, 2007.

[146] A. Jsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.

[147] L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, number 2870 in Lecture Notes in Computer Science, pages 402–418. Springer Berlin Heidelberg, Jan. 2003.

[148] L. Kagal, T. Finin, M. Paolucci, N. Srinivasan, K. Sycara, and G. Denker. Authorization and privacy for semantic web services. *Intelligent Systems, IEEE*, 19(4):50–56, 2004.

[149] M. Kaminsky and E. Banks. SFS-HTTP: Securing the web with self-certifying URLs, 1999.

[150] H. Kamoda, M. Yamaoka, S. Matsuda, K. Broda, and M. Sloman. Policy conflict analysis using free variable tableaux for access control in web services environments. In *Proceedings of the Policy Management for the Web Workshop at the 14th International World Wide Web Conference (WWW)*, pages 121–126. Japan, 2005.

[151] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in p2p networks. In *Twelfth International WWW Conference*, Budapest, Hungary, 2003.

[152] D. R. Karger and W. Jones. Data unification in personal information management. *Communications of the ACM*, 49(1):77–82, 2006.

[153] G. Karjoth and M. Schunter. A privacy policy model for enterprises. In *15th Computer Security Foundations Workshop (CSFW'02)*, pages 271–281, Cape Breton, Nova Scotia, Canada,, 2002. IEEE.

[154] G. Karjoth, M. Schunter, and M. Waidner. Platform for enterprise privacy practices: Privacy-enabled management of customer data. In *2ndWorkshop on Privacy Enhancing Technologies*, LNCS. Springer-Verlag, Berlin, 2002.

[155] B. Katt, R. Breu, M. Hafner, T. Schabetsberger, R. Mair, and F. Wozak. Privacy and access control for IHE-based systems. In D. Weerasinghe, editor, *Electronic Healthcare*, number 0001 in Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 145–153. Springer Berlin Heidelberg, Jan. 2009.

[156] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, 13(6):47–55, 1996.

[157] R. Khare and A. Rifkin. Weaving a web of trust. *World Wide Web Journal*, 2(3):77–112, 1997.

[158] Y. Kim and E. Song. Privacy-aware role based access control model: Revisited for multi-policy conflict detection. In *Information Science and Applications (ICISA), 2010 International Conference on*, pages 1–7. IEEE, 2010.

[159] H. C. Koh and G. Tan. Data mining applications in healthcare. *Journal of Healthcare Information ManagementVol*, 19(2):65, 2011.

[160] J. Kovse, T. Hrder, and N. Ritter. *Supporting Mass Customization by Generating Adjusted Repositories for Product Configuration Knowledge.*

[161] J. Krumm. A survey of computational location privacy. *Personal and Ubiquitous Computing*, 13(6):391–399, 2009.

[162] R. Kukafka, J. S. Ancker, C. Chan, J. Chelico, S. Khan, S. Mortoti, K. Natarajan, K. Presley, and K. Stephens. Redesigning electronic health record systems to support public health. *Journal of Biomedical Informatics*, 40(4):398–409, Aug. 2007.

[163] P. Kumaraguru, L. Cranor, J. Lobo, and S. Calo. A survey of privacy policy languages. In *3rd ACM Symposium on Usable Privacy and Security*, 2007.

[164] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[165] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[166] D. B. Lange, M. Oshima, G. Karjoth, and K. Kosaka. Aglets: Programming mobile agents in java. In T. Masuda, Y. Masunaga, and M. Tsukamoto, editors, *Worldwide Computing and Its Applications*, number 1274 in Lecture Notes in Computer Science, pages 253–266. Springer Berlin Heidelberg, Jan. 1997.

[167] M. Li, X.-Y. Du, and S. Wang. Learning ontology from relational database. In *Proceedings of 2005 International Conference on Machine Learning and Cybernetics, 2005*, volume 6, pages 3410–3415 Vol. 6, 2005.

[168] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 106–115, 2007.

[169] N. Li and J. Mitchell. RT: a role-based trust-management framework. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 201–212 vol.1, 2003.

[170] Z. Li, X. Xu, L. Shi, J. Liu, and C. Liang. Authentication in peer-to-peer network: Survey and research directions. In *Network and System Security, 2009. NSS'09. Third International Conference on*, pages 115–122, 2009.

[171] H. Liang, W. Sun, X. Zhang, and Z. Jiang. A policy framework for collaborative web service customization. In *Service-Oriented System Engineering, 2006. SOSE'06. Second IEEE International Workshop*, pages 197–204, 2006.

[172] H. Liu, X. Hou, G. Hu, J. Li, and Y. Q. Ding. Development of an EHR system for sharing-a semantic perspective. In *MIE*, pages 113–117, 2009.

[173] X. Liu, Y. Hui, W. Sun, and H. Liang. Towards service composition based on mashup. In *Services, 2007 IEEE Congress on*, pages 332–339. IEEE, 2007.

[174] H. Lockhart, S. Andersen, J. Bohren, Y. Sverdlov, M. Hondo, H. Maruyama, A. Nadalin, N. Nagaratnam, T. Boubez, and K. Morrison. Web services federation language (WS-federation). *Web services security specification*, 2006.

[175] J. Lopez, R. Oppliger, and G. Pernul. Authentication and authorization infrastructures (AAIs): a comparative survey. *Computers & Security*, 23(7):578–590, 2004.

[176] J. Lu, L. Ma, L. Zhang, J.-S. Brunner, C. Wang, Y. Pan, and Y. Yu. SOR: a practical system for ontology storage, reasoning and search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1402–1405, 2007.

[177] R. MacGregor. Inside the loom description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[178] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[179] P. C. D. O. Malaysia. PCDOM PrimaCare.http://www.primacare.org.my.

[180] S. Malek, N. Beckman, M. Mikic-Rakic, and N. Medvidovic. A framework for ensuring and improving dependability in highly distributed systems. In *Architecting Dependable Systems III*. Springer Verlag, 2005.

[181] D. J. Mandell and S. A. McIlraith. A bottom-up approach to automating web service discovery, customization, and semantic translation. In *The Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web (ESSW'03). Budapest*, 2003.

[182] N. Mangano, A. Baker, and A. van der Hoek. Calico: A prototype sketching tool for modeling in early design. In *Second International Workshop on Modeling in Software Engineering (MiSE 2008), held in conjunction with ICSE 2008*, Leipzig, Germany, 2008.

[183] C. Martinez-Cruz, I. J. Blanco, and M. A. Vila. Ontologies versus relational databases: are they so different? a comparison. *Artificial Intelligence Review*, 38(4):271–290, Dec. 2012.

[184] L. Martino and S. Ahuja. Privacy policies of personal health records: an evaluation of their effectiveness in protecting patient information. In *Proceedings of the 1st ACM International Health Informatics Symposium*, pages 191–200, 2010.

[185] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 413–427. IEEE, 2012.

[186] C. J. McDonald, S. M. Huff, J. G. Suico, G. Hill, D. Leavelle, R. Aller, A. Forrey, K. Mercer, G. DeMoor, and J. Hook. LOINC, a universal standard for identifying laboratory observations: a 5-year update. *Clinical chemistry*, 49(4):624–633, 2003.

[187] S. H. . C. S. Medicine. Patient privacy.

[188] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26:70–93, 2000.

[189] N. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. In *Proceedings of the 22nd international conference on Software engineering*, pages 178–187. ACM, 2000.

[190] N. Milanovic and M. Malek. Current solutions for web service composition. *Internet Computing, IEEE*, 8(6):51–59, 2004.

[191] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. *Peer-to-peer computing*. Technical Report HPL-2002-57, HP Lab, 2002.

[192] M. C. Mont, S. Pearson, and P. Bramhall. Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services. In *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pages 377–382, 2003.

[193] M. E. Morton and S. Wiedenbeck. A framework for predicting EHR adoption attitudes: A physician survey. *Perspectives in Health Information Management / AHIMA, American Health Information Management Association*, 6(Fall), Sept. 2009.

[194] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist. WS-trust 1.4, 2009.

[195] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web services security: SOAP message security 1.0 (WS-security 2004). *Oasis Standard*, 200401:1–20010502, 2004.

[196] M. Naedele. Standards for XML and web services security. *Computer*, 36(4):96–98, 2003.

[197] S. Narayan, M. Gagn, and R. Safavi-Naini. Privacy preserving EHR system using attribute-based infrastructure. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 47–52, New York, NY, USA, 2010. ACM.

[198] A. Narayanan and V. Shmatikov. Myths and fallacies of personally identifiable information. *Communications of the ACM*, 53(6):24–26, 2010.

[199] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[200] W. Nejdl, D. Olmedilla, and M. Winslett. Peertrust: Automated trust negotiation for peers on the semantic web. In *Secure Data Management*, pages 118–132. Springer, 2004.

[201] T. Nguyen and A. Colman. A feature-oriented approach for web service customization. In *2010 IEEE International Conference on Web Services (ICWS)*, pages 393–400, July 2010.

[202] OASIS. Cross-enterprise security and privacy authorization (XSPA) profile of XACML v2.0 for healthcare version. OASIS standard, OASIS, Nov. 2009.

[203] OASIS. eXtensible access control markup language (XACML) version 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf, 2013.

[204] L. O'Brien, P. Merson, and L. Bass. Quality attributes for service-oriented architectures. In *Proceedings of the international Workshop on Systems Development in SOA Environments*, page 3, 2007.

[205] U. of Louisville Hospital. Notice of privacy policy university hospital - UofL health care.

[206] U. S. D. of Veteran Affairs. VistA - eHealth.

[207] C. Okasaki. *Purely functional data structures*. Cambridge University Press, 1999.

[208] W. H. Organization and others. International classification of diseases (ICD), 2012.

[209] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[210] J. Park and R. Sandhu. Towards usage control models: beyond traditional access control. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, SACMAT '02, pages 57–64, New York, NY, USA, 2002. ACM.

[211] J. S. PARK, R. SANDHU, and G.-J. AHN. Role-based access control on the web. *ACM Transactions on Information and System Security*, 4(1):37–71, 2001.

[212] M. Peleg, D. Beimel, D. Dori, and Y. Denekamp. Situation-based access control: Privacy management via modeling of patient data access scenarios. *Journal of Biomedical Informatics*, 41(6):1028–1040, Dec. 2008.

[213] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[214] PicketLink. PicketLink token registry.

[215] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *11th World Wide Web Conference (Web Engineering Track)*, pages 7–11, 2002.

[216] W. Pratt, K. Unruh, A. Civan, and M. M. Skeels. Personal health information management. *Communications of the ACM*, 49(1):51–55, 2006.

[217] C. A. Puri, K. Gomadam, P. Jain, P. Z. Yeh, and K. Verma. Multiple ontologies in healthcare information technology: Motivations and recommendation for ontology mapping and alignment. In *ICBO*, 2011.

[218] T. Quatrani. *Visual Modeling With Rational Rose and UML*. Addison-Wesley, Reading, MA, 1997.

[219] V. Rafe and M. Hajvali. Designing an architectural style for pervasive healthcare systems. *Journal of medical systems*, 37(2):9927, 2013.

[220] S. Raghavan and H. Garcia-Molina. Integrating diverse information management systems: A brief survey. Technical report, Stanford University, 2001.

[221] M. Raunak and L. Osterweil. Resource management for complex, dynamic environments. *IEEE Transactions on Software Engineering*, 39(3):384–402, Mar. 2013.

[222] A. Rezgui, M. Ouzzani, A. Bouguettaya, and B. Medjahed. Preserving privacy in web services. In *4th international workshop on Web information and data management*, pages 56–62, 2002.

[223] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2007.

[224] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, number 2870 in Lecture Notes in Computer Science, pages 351–368. Springer Berlin Heidelberg, Jan. 2003.

[225] T. C. Rindfleisch. Privacy, information technology, and health care. *Communications of the ACM*, 40(8):92–100, 1997.

[226] J. Sabater and C. Sierra. Review on computational trust and reputation models. *Artificial Intelligence Review*, 24(1):33–60, 2005.

[227] J. H. Saltzer. Protection and the control of information sharing in multics. *Communications of the ACM*, 17(7):388–402, 1974.

[228] Y. Sam, O. Boucelma, and M.-S. Hacid. Web services customization: a composition-based approach. In *Proceedings of the 6th international conference on Web engineering*, pages 25–31, 2006.

[229] P. Samarati and S. C. d. Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, number 2171 in Lecture Notes in Computer Science, pages 137–196. Springer Berlin Heidelberg, Jan. 2001.

[230] K. Samuel, L. Obrst, S. Stoutenberg, K. Fox, P. Franklin, A. Johnson, K. Laskey, D. Nichols, S. Lopez, and J. Peterson. Translating owl and semantic web rules into prolog: Moving toward description logic programs. *Theory and Practice of Logic Programming*, 8(3):301–322, 2008.

[231] R. S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.

[232] M. B. Sanrom, A. Valls, and K. Gibert. *Survey of Electronic Health Record Standards*. Universitat Rovira i Virgili. Departament d'Enginyeria Informtica, 2006.

[233] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measurement study of peer-to-peer file sharing systems. In *Electronic Imaging 2002*, pages 156–170, 2001.

[234] P. Schloeffel, T. Beale, G. Hayworth, S. Heard, and H. Leslie. The relationship between CEN 13606, HL7, and openEHR. *HIC 2006 and HINZ 2006: Proceedings*, 24, 2006.

[235] M. Shaw and D. Garlan. *Software Architecture: Perpectives on an Emerging Discipline*. Prentice Hall, 1996.

[236] A. Singh and P. Anand. State of art in ontology development tools. *International Journal*, 2(7), 2013.

[237] E. G. Sirer and K. Wang. An access control language for web services. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 23–30. ACM, 2002.

[238] B. Sobolev, D. Harel, C. Vasilakis, and A. Levy. Using the statecharts paradigm for simulation of patient flow in surgical care. *Health care management science*, 11(1):79–86, Mar. 2008.

[239] K. A. Spackman, K. E. Campbell, and R. A. Ct. SNOMED RT: a reference terminology for health care. *Proceedings: a conference of the American Medical Informatics Association / ... AMIA Annual Fall Symposium. AMIA Fall Symposium*, pages 640–644, 1997.

[240] B. Srivastava and J. Koehler. Web service composition-current solutions and open problems. In *ICAPS 2003 workshop on Planning for Web Services*, volume 35, pages 28–35, 2003.

[241] O. Standard. *WS-SecureConversation 1.4*. February, 2009.

[242] O. Standard. *WS-SecurityPolicy 1.3*. February, 2009.

[243] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Winter*, pages 191–202, 1988.

[244] R. Stevens, C. A. Goble, and S. Bechhofer. Ontology-based knowledge representation for bioinformatics. *Briefings in Bioinformatics*, 1(4):398–414, Jan. 2000.

[245] M. Stollberg and M. Muth. Service customization by variability modeling. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 425–434, 2010.

[246] W. V. Sujansky, S. A. Faus, E. Stone, and P. F. Brennan. A method to implement fine-grained access control for personal health records through standard relational database queries. *Journal of Biomedical Informatics*, 43(5, Supplement):S46–S50, Oct. 2010.

[247] W. Sun, X. Zhang, C. J. Guo, P. Sun, and H. Su. Software as a service: Configuration and customization perspectives. In *Congress on Services Part II, 2008. SERVICES-2. IEEE*, pages 18–25, 2008.

[248] G. Suryanarayana, M. H. Diallo, J. R. Erenkrantz, and R. N. Taylor. Architectural support for trust models in decentralized applications. In *28th International Conference on Software Engineering (ICSE 2006)*, pages 52–61, Shanghai, China, 2006.

[249] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness, and Knowledge-based Systems*, 10(5):557–570, 2002.

[250] A. S. Tanenbaum. *Computer Networks, 4-th Edition*. Prentice Hall, 2003.

[251] P. C. Tang, J. S. Ash, D. W. Bates, J. M. Overhage, and D. Z. Sands. Personal health records: definitions, benefits, and strategies for overcoming barriers to adoption. *Journal of the American Medical Informatics Association*, 13(2):121–126, 2006.

[252] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software architecture: foundations, theory, and practice*. Wiley Publishing, 2009.

[253] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, rei, and ponder. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, number 2870 in Lecture Notes in Computer Science, pages 419–437. Springer Berlin Heidelberg, Jan. 2003.

[254] A. Tumer, A. Dogac, and I. H. Toroslu. A semantic-based user privacy protection framework for web services. In B. Mobasher and S. S. Anand, editors, *Intelligent Techniques for Web Personalization*, number 3169 in Lecture Notes in Computer Science, pages 289–305. Springer Berlin Heidelberg, Jan. 2005.

[255] I. University of California. Graduate student health insurance plan (GSHIP), 2013.

[256] M. Uschold, M. Gruninger, M. Uschold, and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136, 1996.

[257] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, pages 93–96, 2003.

[258] W. Van Der Aalst and A. Ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[259] H. van der Linden, D. Kalra, A. Hasman, and J. Talmon. Inter-organizational future proof EHR systems: A review of the security and privacy related issues. *International Journal of Medical Informatics*, 78(3):141–160, 2009.

[260] von Riegen, Claus and Trickovic, Ivana. Using BPEL4ws in a UDDI registry.

[261] W3C. Web services choreography description language: Primer, June 2006.

[262] Y. Wang and A. Adviser-Kobsa. *A framework for Privacy-Enhanced personalization*. California State University at Long Beach, 2010.

[263] Y. Wang and J. Vassileva. Trust and reputation model in peer-to-peer networks. In *Peer-to-Peer Computing, 2003.(P2P 2003). Proceedings. Third International Conference on*, pages 150–157, 2003.

[264] P. Wohed, W. M. van der Aalst, M. Dumas, and A. H. ter Hofstede. Analysis of web services composition languages: The case of bpel4ws. In *Conceptual Modeling-ER 2003*, pages 200–215. Springer, 2003.

[265] WONCA International Classification Committee. *ICPC-2: International Classification of Primary Care*. Oxford University Press, 1998.

[266] J. Wong and J. I. Hong. Making mashups with marmite: Towards end-user programming for the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1435–1444, New York, NY, 2007. ACM.

[267] World Health Organization. The anatomical therapeutic chemical classification system with defined daily doses (ATC/DDD), 2003.

[268] WorldVistA.org. WorldVistA. http://www.worldvista.org/.

[269] J. Xiao, L. J. Osterweil, and Q. Wang. Dynamic scheduling of emergency department resources. In *Proceedings of the 1st ACM International Health Informatics Symposium*, pages 590–599. ACM, 2010.

[270] Y. Xu, K. Wang, B. Zhang, and Z. Chen. Privacy-enhancing personalized web search. In *Proceedings of the 16th international conference on World Wide Web*, pages 591–600, 2007.

[271] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 379–390, 2004.

[272] S. Ye, F. Wu, R. Pandey, and H. Chen. Noise injection for search privacy protection. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 3, pages 1–8, 2009.

[273] G. Yee and L. Korba. Privacy policy compliance for web services. In *IEEE International Conference on Web Services (ICWS 2004)*, San Diego, CA, 2004. IEEE.

[274] J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, 2008.

[275] E. Yuan and J. Tong. Attributed based access control (ABAC) for web services. In *IEEE International Conference on Web Services*, 2005.

[276] L. Zhang, G.-J. Ahn, and B.-T. Chu. A rule-based framework for role based delegation. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, SACMAT '01, pages 153–162, New York, NY, USA, 2001. ACM.

[277] L. Zhang, G.-J. Ahn, and B.-T. Chu. A role-based delegation framework for healthcare information systems. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, SACMAT '02, pages 125–134, New York, NY, USA, 2002. ACM.

[278] R. Zhang and L. Liu. Security models and requirements for healthcare application clouds. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 268–275, 2010.

Appendix A

Examples of privacy policies

Following, we provide a collection of sample health care practitioners' privacy policies which illustrate the type of authorization and rights parties have in this domain and that ought to be captured to offer policy-based services. These policies have been obtained from hospitals' and insurance companies' publicly available privacy policy documents, and from scenarios found in the academic literature. We differentiate between internal and external policies: internal policies are those mainly for the purpose of division of labour among an organization's employees, while external policies communicate how patient's data will be handled and the conditions under which patient data is disclosed.

A.0.1 Hospital organizational privacy policies

Internal policies

- We may collect information about the patient from different sources, such as from the patient himself/herself, his/her family or designated representative, insurance companies, employers or other health care providers. Such information may be name, address,

birth date, social security number, medical and mental health history, payment sources, physicians names, family contacts, and medical information such as test results, health records, and diagnosis.

- We may contact the patient as a reminder of an appointment for care or treatment at our facility.
- We may contact the patient about treatment options, health-related benefits, or other products or services that may be of interest.
- We may also contact you to conduct case management or care coordination.
- Only authorized doctors can see or update certain patients data (e.g., only clinical psychologists can update the patient's psychological evaluation).
- Only a specialist doctor may be allowed to see a section of the records of his/her patient that pertain to the results of very sensitive medical test.
- We may share patient information with our departments to coordinate the different health care services that he/she may need, such as prescriptions, lab work, or x-rays.
- No-one shall have the ability to delete clinical information until the appropriate time period has expired (6 years).
- All accesses to clinical records shall be marked on the record with the subjects name, as well as the date and time.
- A doctor should be allowed to access a patient's health record only when the patient has designated him as his/her primary care physician.
- Only a specialist physician (e.g. cardiologist) may prescribe drugs for illnesses related to his/her specialty (e.g. cardiac-related).
- Only physicians may order lab tests.

- Only members of a patients active care team should be able to access to the patients records. Thus, although a physician may have the right to order a lab test by virtue of the qualifications and responsibilities that determine his role, the physician should have the right to do so for the patient's record only when he/she is part of the patient's care team. Another example, the primary care team in general wards should be given access to a patients records only after the ER unit has requested transfer of the patient to one of those wards.

- Certain team members may delegate duties and associated permissions to other team members.

- The primary care physician in care of the patient is provided with read/write access to a prescription file.

- The hospital pharmacists have only read access to prescription files.

External policies

- Patient may ask us to restrict how we use and disclose his/her information to carry out treatment, payment and health care operations to your family, friends or other persons you identify. We may agree to or deny your request.

- Patient health information is not to be disclosed for any purpose unless the patient has signed a form authorizing the use or disclosure.

- The patient may revoke a given authorization in writing, unless we have taken any action in reliance on the authorization (interpreted as "authorization revocation cannot be retroactive").

- The patient has the right to inspect and obtain a copy of the medical records that the Hospital uses, subject to certain limited exceptions. This information includes your

medical and billing records, but may not include some mental health information.

- We may disclose protected health information as necessary with health care providers involved in your treatment, such as physicians, hospital staff or outside consultants (e.g. pathologists or radiologists) or for the purpose of making decisions about patients' care to a doctor or health facility upon request.

- We may share patient information with other health care facilities to which he/she is transferred.

- We will make uses and disclosures of patients' medical procedures and treatment and submit claims to health insurance companies for billing purposes, verify eligibility, or to request pre-authorization of medical services.

- We may send health plan statements to the policyholder, who may or may not be the patient.

- With patient's approval, we may disclose your protected health information to designated family, friends, and others who are involved in your care.

- We may share limited protected health information with designated family, friends, and others who are involved in their care without your approval if the patient is unavailable, incapacitated, or facing an emergency medical situation.

- We may disclose limited protected health information to a public or private entity that is authorized to assist in disaster relief efforts in order for that entity to locate a family member or other persons that may be involved in some aspect of caring for the patient.

- We may use a patient's' protected health information to determine whether he/she might be interested in or benefit from treatment alternatives or other health-related programs, products or services which may be available to you as a member of the health plan.

- We may share patient information with public health or other legal authorities charged with preventing or controlling disease, injury or disability (e.g. CDC or state public health departments).

- We may share patient information with a person who may have been exposed to a communicable disease or may otherwise be at risk for contracting or spreading a disease or condition when permitted by law.

- We may share patient information as authorized under workers compensation laws or other similar programs established by law.

- We may share patient information to conduct our Academic Medical Center training programs, such as for physicians who are pursuing advanced training or faculty and students of academic health affiliates.

- We may share patient information for medical research when approved by an Institutional Review Board (IRB) or a Privacy Board. For example, we may compile research databases or create limited data sets permitted by federal regulations.

- We may share patient information with the Food and Drug Administration (FDA) to support activities related to the quality, safety or effectiveness of a product or activity.

- We may share patient information to compile patient census data, conduct quality improvement programs, or review the qualifications of health care professionals.

- We may share patient information as required by law to report abuse or domestic violence.

- We may share patient information as required by law in response to a warrant, subpoena or summons.

- We may share patient information as required by law to identify or locate suspects, witnesses or missing persons.

- We may share patient information as required by law if patient is, or is suspected of being, a victim of a crime
- We may share patient information as required by law to alert law enforcement officials when we believe a death may have resulted from criminal conduct.
- We may also share patient information with law enforcement authorities to identify or apprehend an individual.
- We may share patient information as required by military command authorities if the patient is a member of the Armed Forces.
- The patient may receive an accounting of disclosures of your information made by us in the six (6) years prior to the date of request.
- Patient may inspect and obtain a copy of his/her protected health information contained in a designated record set for as long as the information is maintained.

A.0.2 Insurance company organizational privacy policies

- We may use information regarding patients' medical procedures and treatment to process and pay claims.
- We may use information regarding patients' medical procedures and treatment to determine whether services are medically necessary.
- We may use information regarding patients' medical procedures and treatment to otherwise pre-authorize or certify services as covered under your health benefits plan.
- We may forward patients' medical procedures and treatment information to another health plan, which may also have an obligation to process and pay claims on your behalf.

- We may disclose patients' protected health information to another health care facility, health care professional, or health plan for quality assurance and case management, but only if that facility, professional, or plan also has or had a patient relationship with you.

- With patient's approval, we may disclose your protected health information to designated family, friends, and others who are involved in payment for your care in order to facilitate that persons paying for your care.

- If the patient has designated a person to receive information regarding payment of the premium on your long-term care or Medicare supplemental policy, we will inform that person when the premium has not been paid.

- At times it may be necessary for us to provide some of your protected health information outside persons or organizations, such as auditing, accreditation, actuarial services, legal services who assist us with our health care operations.

- We may communicate with the patient regarding his/her claims, premiums, or other things connected with your health plan or insurance.

- The patient has the right to request to receive communications regarding your protected health information from us by alternative means or at alternative locations (if a patient wishes messages to not be left on voice mail or sent to a particular address, we will accommodate reasonable requests).

- We may request and receive from the patient and his/her health care providers protected health information either prior to enrollment in the health plan to determine eligibility to enroll and to determine rates.

Appendix B

Sample healthcare policies expressed in various policy languages

In section 5.2 we evaluated a set policy the languages for their ability to express organizational privacy and operational policies in the healthcare domain. To do so, we selected a set of sample policies described in table 5.1 and formalized them using the evaluated languages. These policies where selected (and modified as required to fit the evaluation criteria) from the domain policies in appendix A. Evaluation criteria included ability to express subjects, their active roles, objects, attributes, obligations, rights, prohibitions, conditions, exceptions, usage (purpose), and temporal constraints.

Following, we provide the specification of the policies in table 5.1 in EPAL, XACML, Cassandra, PeerTrust, Ponder, and Rei (all policies are specified in all languages).

B.1 EPAL

Policy Document Domain Vocabulary

```
<epal-vocabulary version="1.2" xmlns="http://www.research.ibm.com/
   privacy/epal" xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
   schemaLocation="http://www.research.ibm.com/privacy/epal epal.
   xsd http://www.w3.org/2001/XMLSchema xs-dummy.xsd ">
  <vocabulary-information id="EHR-vocabulary">
   <short-description language="en">EHR vocabulary</
   short-description>
       <long-description language="en">Vocabulary for the
   implementation of the policies governing the EHR system</
   long-description>
    <issuer>
          <name>Administrative department</name>
            <organization>Hospital ABC</organization>
            <e-mail>admin@hospitalabc.com</e-mail>
            <address>Sample address</address>
            <country>United States</country>
      </issuer>
        <location>http://www.hospitalabc.com</location>
    <version-info end-date="2001-12-31T12:00:00" last-modified="
   2001-12-31T12:00:00"  revision-number="1" start-date="
   2001-12-31T12:00:00" test="false"/>
  </vocabulary-information>
```

```
  <user-category id="Person">
      <short-description language="en">Person</short-description
 >
      <long-description language="en">Person</long-description>
  </user-category>

<user-category id="Staff" parent="Person">
      <short-description language="en">Hospital staff</
 short-description>
      <long-description language="en">Hospital staff</
 long-description>
  </user-category>

<user-category id="Physician" parent="Staff">
      <short-description language="en">Physicians</
 short-description>
      <long-description language="en">Physicians employees of
 the Hospital ABC</long-description>
  </user-category>

<user-category id="PCP" parent="Physician">
      <short-description language="en">PCP</short-description>
      <long-description language="en">Primary care physician</
 long-description>
  </user-category>

<user-category id="Cardiologists" parent="Physician">
      <short-description language="en">Cardiologists</
 short-description>
```

```
      <long-description language="en">Physicians specialists in
 cardiology</long-description>
  </user-category>

<user-category id="Patient">
      <short-description language="en">Patient</
 short-description>
      <long-description language="en">Patient treated in the
 Hospital ABC</long-description>
  </user-category>

<user-category id="Organization">
      <short-description language="en">Organization</
 short-description>
      <long-description language="en">External organization</
 long-description>
  </user-category>

 <data-category id="EHR">
      <short-description language="en">EHR</short-description>
      <long-description language="en">Electronic Health Records
 </long-description>
  </data-category>

<data-category id="AnonymousEHR">
      <short-description language="en">AnonymousEHR</
 short-description>
      <long-description language="en">Anonymous Electronic
 Health Records</long-description>
```

```
  </data-category>

<data-category id="CardEHR" parent="EHR">
      <short-description language="en">Cardiology EHR</
 short-description>
      <long-description language="en">Cardiology related
 information within a patient's Electronic Health Records</
 long-description>
  </data-category>

<data-category id="Prescriptions">
      <short-description language="en">Prescriptions</
 short-description>
      <long-description language="en">List of drugs prescribed
 to a patient</long-description>
  </data-category>

<data-category id="LabOrders">
      <short-description language="en">Laboratory Orders</
 short-description>
      <long-description language="en">Laboratory Orders</
 long-description>
  </data-category>

<purpose id="Treatment">
     <short-description language="en">Medical treatment</
 short-description>
      <long-description language="en">Medical treatment</
 long-description>
```

```
  </purpose>

<purpose id="AdministrativeHousekeeping">
     <short-description language="en">Administrative
 housekeeping</short-description>
      <long-description language="en">Administrative information
  organization</long-description>
  </purpose>

<purpose id="Research">
     <short-description language="en">Research</
 short-description>
      <long-description language="en">Research</long-description
 >
  </purpose>

<purpose id="Supervision">
     <short-description language="en">Supervision</
 short-description>
      <long-description language="en">Supervision</
 long-description>
  </purpose>

 <action id="Access">
      <short-description language="en">Access</short-description
 >
      <long-description language="en">Access which includes
 viewing and modifying</long-description>
  </action>
```

```
<action id="Delete">
      <short-description language="en">Delete</short-description
 >
      <long-description language="en">Delete</long-description>
  </action>

<action id="Prescribe">
      <short-description language="en">Prescribe drugs</
 short-description>
      <long-description language="en">Prescribe drugs</
 long-description>
  </action>

<action id="Read">
      <short-description language="en">Read</short-description>
      <long-description language="en">Read data only</
 long-description>
  </action>

<action id="Order">
      <short-description language="en">Order</short-description>
      <long-description language="en">Order</long-description>
  </action>

<container id="Person">
    <short-description>Person details</short-description>
  <attribute id="Name" maxOccurs="1" minOccurs="1" simpleType="
 http://www.w3.org/2001/XMLSchema#string">
```

```
        <short-description>Person's full name</short-description
 >
   </attribute>
</container>

<container id="Physician">
    <short-description>Physician details</short-description>
  <attribute id="Name" maxOccurs="1" minOccurs="1" simpleType="
 http://www.w3.org/2001/XMLSchema#string">
        <short-description>Physician's full name</
 short-description>
   </attribute>
   <attribute id="Speciality" maxOccurs="unbounded" minOccurs="1
 " simpleType="http://www.w3.org/2001/XMLSchema#string">
        <short-description>Physician's speciality</
 short-description>
   </attribute>
</container>

<container id="EHR">
    <short-description>Electronic health record</
 short-description>
    <attribute id="ModificationDate" maxOccurs="1" minOccurs="1"
  simpleType="http://www.w3.org/2001/XMLSchema#dateTime">
        <short-description>Record modification date</
 short-description>
   </attribute>
</container>
```

```
<container id="Patient">
    <short-description>Patient details</short-description>
  <attribute id="TreatedCondition" maxOccurs="unbounded"
 minOccurs="1" simpleType="http://www.w3.org/2001/XMLSchema#
 string">
        <short-description>Condition or illness treated</
 short-description>
   </attribute>
   <attribute id="Age" maxOccurs="1" minOccurs="1" simpleType="
 http://www.w3.org/2001/XMLSchema#integer">
        <short-description>Patient's age</short-description>
   </attribute>
   <attribute id="TreatedConditionArea" maxOccurs="unbounded"
 minOccurs="1" simpleType="http://www.w3.org/2001/XMLSchema#
 string">
        <short-description>Medical area to which the condition
 or illness treated belongs</short-description>
   </attribute>
    <attribute id="PCP" maxOccurs="1" minOccurs="0" simpleType="
 http://www.w3.org/2001/XMLSchema#string">
        <short-description>Name of patient's primary care
 physician</short-description>
   </attribute>
     <attribute id="CareMember" maxOccurs="unbounded" minOccurs=
 "1" simpleType="http://www.w3.org/2001/XMLSchema#string">
        <short-description>Physician or care provider on patient
 's active care</short-description>
   </attribute>
```

```
      <attribute id="Guardian" maxOccurs="0" minOccurs="2"
    simpleType="http://www.w3.org/2001/XMLSchema#string">
          <short-description>Patient's parent or legal guardian</
    short-description>
      </attribute>
  </container>

  <container id="Organization">
      <short-description>Organization details</short-description>
     <attribute id="Name" maxOccurs="1" minOccurs="1" simpleType="
    http://www.w3.org/2001/XMLSchema#string">
          <short-description>Name</short-description>
      </attribute>
      <attribute id="authorized" maxOccurs="1" minOccurs="1"
    simpleType="http://www.w3.org/2001/XMLSchema#boolean">
          <short-description>Authorized?</short-description>
      </attribute>
  </container>
</epal-vocabulary>


<epal-policy default-ruling="allow" global-condition="NCName"
    version="1.2" xmlns="http://www.research.ibm.com/privacy/epal"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
    www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http
    ://www.research.ibm.com/privacy/epal epal.xsd http://www.w3.org
    /2001/XMLSchema xs-dummy.xsd ">
  <policy-information id="Hospital policy">
```

```
     <short-description language="en">Privacy policy Hospital
ABC</short-description>
 <long-description language="en">Privacy policy Hospital ABC</
long-description>
 <issuer>
       <name>Administrative department</name>
         <organization>Hospital ABC</organization>
         <e-mail>admin@hospitalabc.com</e-mail>
         <address>Sample address</address>
         <country>United States</country>
   </issuer>
 <location>http://www.hospitalabc.com</location>
     <version-info end-date="2001-12-31T12:00:00" last-modified
="2001-12-31T12:00:00"  revision-number="" start-date="
2001-12-31T12:00:00" test="false"/>
 </policy-information>
<epal-vocabulary-ref id="EHR-vocabulary" location="http://www.
hospitalabc.com" revision-number="1"/>

<condition id="Cardiology?">
 <predicate refid="http://www.research.ibm.com/privacy/epal#
string-equal">
         <function refid="http://www.research.ibm.com/privacy/
epal#string-bag-to-value">
           <attribute-reference container-refid="Physician"
attribute-refid="Speciality"/>
         </function>
         <attribute-value simpleType="http://www.w3.org/2001/
XMLSchema#string">Cardiology</attribute-value>
```

```
      </predicate>
  </condition>

   <condition id="Older_6_years?">
   <predicate refid="http://www.research.ibm.com/privacy/epal#
  integer-greater-than">
           <function refid="http://www.research.ibm.com/privacy/
  epal#string-bag-to-value">
             <attribute-reference container-refid="EHR"
  attribute-refid="\textcolor{red}{?}"/>
           </function>
           <attribute-value simpleType="http://www.w3.org/2001/
  XMLSchema#integer">6</attribute-value>
      </predicate>
  </condition>

 <condition id="PCP?">
   <predicate refid="http://www.research.ibm.com/privacy/epal#
  string-equal">
           <function refid="http://www.research.ibm.com/privacy/
  epal#string-bag-to-value">
             <attribute-reference container-refid="Patient"
  attribute-refid="PCP"/>
           </function>
           <function refid="http://www.research.ibm.com/privacy/
  epal#string-bag-to-value">
             <attribute-reference container-refid="Physician"
  attribute-refid="Name"/>
           </function>
```

```
     </predicate>
</condition>

<condition id="AuthorityToPrescribe?">
 <predicate refid="http://www.research.ibm.com/privacy/epal#
string-equal">
         <function refid="http://www.research.ibm.com/privacy/
epal#string-bag-to-value">
           <attribute-reference container-refid="Physician"
attribute-refid="Speciality"/>
         </function>
         <function refid="http://www.research.ibm.com/privacy/
epal#string-bag-to-value">
           <attribute-reference container-refid="Patient"
attribute-refid="TreatedConditionArea"/>
         </function>
     </predicate>
</condition>

<condition id="Authorized?">
 <predicate refid="http://www.research.ibm.com/privacy/epal#
string-equal">
         <function refid="http://www.research.ibm.com/privacy/
epal#string-bag-to-value">
           <attribute-reference container-refid="Organization"
attribute-refid="authorized"/>
         </function>
         <attribute-value simpleType="http://www.w3.org/2001/
XMLSchema#boolean">true</attribute-value>
```

```
      </predicate>
  </condition>

   <condition id="InCareTeam?">
   <predicate refid="http://www.research.ibm.com/privacy/epal#
  string-is-in">
           <function refid="http://www.research.ibm.com/privacy/
  epal#string-bag-to-value">
      <attribute-reference container-refid="Patient"
  attribute-refid="CareMember"/>
         </function>
     <function refid="http://www.research.ibm.com/privacy/epal#
  string-bag-to-value">
            <attribute-reference container-refid="Physician"
  attribute-refid="Name"/>
           </function>
      </predicate>
  </condition>

 <condition id="Guardian?">
   <predicate refid="http://www.research.ibm.com/privacy/epal#
  string-is-in">
           <function refid="http://www.research.ibm.com/privacy/
  epal#string-bag-to-value">
      <attribute-reference container-refid="Patient"
  attribute-refid="Guardian"/>
         </function>
     <function refid="http://www.research.ibm.com/privacy/epal#
  string-bag-to-value">
```

```
            <attribute-reference container-refid="Person"
attribute-refid="Name"/>
          </function>
      </predicate>
</condition>
```

P1: Only cardiologists are allowed to access cardiac medical records.

```
 <rule id="P1" ruling="allow">
       <short-description language="en">P1</short-description>
       <long-description language="en">Only cardiologists are
allowed to access cardiac medical records.</long-description>
       <user-category refid="Cardiologists"/>
       <data-category refid="CardEHR"/>
       <purpose refid="Treatment"/>
       <action refid="Access"/>
       <condition refid="Cardiologist?"/>
 </rule>
```

P2: No one shall have the ability to delete clinical information until the appropriate time period has expired (6 years).

```
 <rule id="P2" ruling="allow">
       <short-description language="en">P2</short-description>
       <long-description language="en">No one shall have the
ability to delete clinical information until the appropriate
time period has expired (6 years).</long-description>
       <user-category refid="Staff"/>
```

```
        <data-category refid="EHR"/>
        <purpose refid="AdministrativeHousekeeping"/>
        <action refid="Delete"/>
        <condition refid="Older_6_years?"/>
    </rule>
```

P3: A doctor should be granted the permissions assigned to the primary care physician of a patient (read and modify his/her EHR) only when the patient has designated him as the primary care physician.

```
    <rule id="P3" ruling="allow">
        <short-description language="en">P3</short-description>
        <long-description language="en">A doctor should be
    granted the permissions assigned to the primary care physician
    of a patient only when the patient has designated him as the
    primary care physician.</long-description>
        <user-category refid="Physician"/>
        <data-category refid="EHR"/>
        <purpose refid="Treatment"/>
        <action refid="Access"/>
        <condition refid="PCP?"/>
    </rule>
```

P4: A specialist physician may prescribe drugs if the treated illnesses is related to his/her specialty.

```
<rule id="P4" ruling="allow">
        <short-description language="en">P4</short-description>
```

```
        <long-description language="en">A specialist physician (
    e.g. cardiologist) may prescribe drugs if the treated illnesses
     is related to his/her specialty (e.g. cardiac-related).</
    long-description>
        <user-category refid="Physician"/>
        <data-category refid="Prescriptions"/>
        <purpose refid="Treatment"/>
        <action refid="Prescribe"/>
        <condition refid="AuthorityToPrescribe?"/>
     </rule>
```

P5: The hospital may limit the access to anonymous medical data only to authorized organizations for research purposes.

```
<rule id="P5" ruling="allow">
        <short-description language="en">P5</short-description>
        <long-description language="en">The hospital may limit
    the access to anonymous medical data only to authorized
    organizations for research purposes.</long-description>
        <user-category refid="Organization"/>
        <data-category refid="AnonymousEHR"/>
        <purpose refid="Research"/>
        <action refid="Read"/>
        <condition refid="Authorzed?"/>
     </rule>
```

P6: Hospital staff other than physicians may not order lab tests.

```
<rule id="P6a" ruling="allow">
        <short-description language="en">P6a</short-description>
```

```
        <long-description language="en">Physicians may order lab
  tests.</long-description>
        <user-category refid="Physitian"/>
        <data-category refid="LabOrders"/>
        <purpose refid="Treatment"/>
        <action refid="Order"/>
  </rule>

<rule id="P6b" ruling="deny">
        <short-description language="en">P6a</short-description>
        <long-description language="en">Hospital staff may not
 order lab tests.</long-description>
        <user-category refid="Staff"/>
        <data-category refid="LabOrders"/>
        <purpose refid="Treatment"/>
        <action refid="Order"/>
  </rule>
```

P7: A physician may order a lab test except when he is not part of the patient's care team.

```
<rule id="P7" ruling="deny">
        <short-description language="en">P7</short-description>
        <long-description language="en">A physician may order a
 lab test except when he is not part of the patients care team
 .</long-description>
        <user-category refid="Physician"/>
        <data-category refid="LabOrders"/>
        <purpose refid="Treatment"/>
```

```
        <action refid="Order"/>
    <condition refid="InCareTeam?"/>
    </rule>
```

P8: A person must be a custodial parent or legal guardian must provide proof of kinship to request access to the electronic health record of a patient under the age of 18.

```
  <rule id="P8" ruling="allow">
        <short-description language="en">P8</short-description>
        <long-description language="en">A custodial parent or
   legal guardian must provide proof of kinship to request access
   to the electronic health record of a patient under the age of
   18.</long-description>
        <user-category refid="Person"/>
        <data-category refid="EHR"/>
        <purpose refid="Supervision"/>
        <action refid="Read"/>
    <condition refid="Guardian?"/>
    </rule>
</epal-policy>
```

B.2 XACML

Policy Document Preamble

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
  xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:
   wd-17 http://docs.oasis-open.org/xacml/3.0/
   xacml-core-v3-schema-wd-17.xsd"
  xmlns:md="http:www.med.example.com/schemas/ehr.xsd"
  xmlns:ct="http:www.med.example.com/schemas/conditionTaxonomy.xsd
   "
  xmlns:ao="http:www.med.example.com/schemas/
   authorizedOrganizations.xsd"
  PolicyId="urn:oasis:names:tc:xacml:3.0:example:EHRPolicy"
       Version="1.0"
  RuleCombiningAlgId="identifier:rule-combining-algorithm:
   deny-overrides">
  <Description> EHR access control policy</Description>
  <PolicyDefaults>
        <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116
   </XPathVersion>
      </PolicyDefaults>
  <Target>
    <AnyOf>
      <AllOf>
```

```
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
anyURI-equal">
        <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#anyURI">
          urn:example:med:schemas:record
        </AttributeValue>
        <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
          AttributeId="urn:oasis:names:tc:xacml:2.0:resource:
target-namespace"
          DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
      </Match>
    </AllOf>
  </AnyOf>
</Target>
```

P1: Only cardiologists are allowed to access cardiac medical records.

```
<Rule RuleId= "urn:oasis:names:tc:xacml:3.0:example:P1" Effect="
Permit">
  <Description>Only cardiologists are allowed to access cardiac
medical records.</Description>
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
```

```
          <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
            cardiologist
          </AttributeValue>
          <AttributeDesignator
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
            AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:role"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:
xpath-node-match">
          <AttributeValue
            DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"
            XPathCategory="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource">
              md:record/md:medical/md:cardiac
          </AttributeValue>
          <AttributeDesignator
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
```

```
          AttributeId="urn:oasis:names:tc:xacml:3.0:
content-selector"
          DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"/>
       </Match>
     </AllOf>
   </AnyOf>
   <AnyOf>
     <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
           read
         </AttributeValue>
         <AttributeDesignator
         MustBePresent="false"
         Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:action"
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:
action-id"
         DataType="http://www.w3.org/2001/XMLSchema#string"/>
       </Match>
     </AllOf>
     <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
```

```
            write
          </AttributeValue>
          <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:3.0:
 attribute-category:action"
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:
 action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
</Rule>
```

P2: No one shall have the ability to delete clinical information until the appropriate time period has expired (6 years).

```
<Rule RuleId= "urn:oasis:names:tc:xacml:3.0:example:P2" Effect="
 Deny">
  <Description>No one shall have the ability to delete clinical
 information until the appropriate time period has expired (6
 years).</Description>
  <Target>
    <AnyOf>
      <AllOf>
```

```
       <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:
xpath-node-match">
         <AttributeValue
           DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"
           XPathCategory="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource">
             md:record
         </AttributeValue>
         <AttributeDesignator
           MustBePresent="false"
           Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
           AttributeId="urn:oasis:names:tc:xacml:3.0:
content-selector"
           DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"/>
       </Match>
     </AllOf>
   </AnyOf>
   <AnyOf>
     <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
           delete
         </AttributeValue>
         <AttributeDesignator
```

```
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:action"
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:
action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
date-less-or-equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
date-one-and-only">
        <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:environment"
          AttributeId="urn:oasis:names:tc:xacml:1.0:environment:
current-date"
          DataType="http://www.w3.org/2001/XMLSchema#date"/>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
date-add-yearMonthDuration">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
:date-one-and-only">
          <AttributeSelector
            MustBePresent="false"
```

```
            Category="urn:oasis:names:tc:xacml:3.0:
 attribute-category:resource"
            Path="md:record/md:modificationDate/text()"
            DataType="http://www.w3.org/2001/XMLSchema#date"/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/
 XMLSchema#yearMonthDuration">
          P6Y
        </AttributeValue>
      </Apply>
    </Apply>
  </Condition>
</Rule>
```

P3: A doctor should be granted the permissions assigned to the primary care physician of a patient (read and modify his/her EHR) only when the patient has designated him as the primary care physician.

```
<Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:P3"
 Effect="Permit">
  <Description>
    A doctor should be granted the permissions assigned to the
 primary care physician of a patient (read and modify his/her
 EHR) only when the patient has designated him as the primary
 care physician.
  </Description>
    <Target>
      <AnyOf>
```

```
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
              physician
            </AttributeValue>
            <AttributeDesignator
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
              AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:role"
              DataType="http://www.w3.org/2001/XMLSchema#string"
/>
          </Match>
        </AllOf>
      </AnyOf>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:
xpath-node-match">
            <AttributeValue
              DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"
              XPathCategory="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource">
                md:record/md:medical
              </AttributeValue>
```

```
            <AttributeDesignator
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
              AttributeId="urn:oasis:names:tc:xacml:3.0:
content-selector"
              DataType="urn:oasis:names:tc:xacml:3.0:data-type
:xpathExpression"/>
         </Match>
       </AllOf>
     </AnyOf>
     <AnyOf>
       <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
              write
           </AttributeValue>
           <AttributeDesignator
             MustBePresent="false"
             Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:action"
             AttributeId="urn:oasis:names:tc:xacml:1.0:action:
action-id"
             DataType="http://www.w3.org/2001/XMLSchema#string"
/>
         </Match>
       </AllOf>
```

```
      </AnyOf>
    </Target>
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
:string-one-and-only">
        <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
          AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:physician-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
:string-one-and-only">
          <AttributeSelector
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
            Path="md:record/md:primaryCarePhysician/md:
registrationID/text()"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
      </Apply>
    </Condition>
  </Rule>
```

P4: A specialist physician may prescribe drugs if the treated illnesses is related to his/her specialty.

```
 <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:P4"
Effect="Permit">
   <Description>
   A specialist physician (e.g. cardiologist) may prescribe
drugs if the treated illnesses is related to his/her specialty
(e.g. cardiac-related).
   </Description>
   <Target>
     <AnyOf>
       <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
             physician
           </AttributeValue>
           <AttributeDesignator
             MustBePresent="false"
             Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
             AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:role"
             DataType="http://www.w3.org/2001/XMLSchema#string"
/>
```

```
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:
xpath-node-match">
          <AttributeValue
            DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"
            XPathCategory="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource">
              md:record/md:medical
            </AttributeValue>
            <AttributeDesignator
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
              AttributeId="urn:oasis:names:tc:xacml:3.0:
content-selector"
              DataType="urn:oasis:names:tc:xacml:3.0:data-type
:xpathExpression"/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
```

```
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
               prescribe
            </AttributeValue>
            <AttributeDesignator
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:action"
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:
action-id"
              DataType="http://www.w3.org/2001/XMLSchema#string"
/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:
any-of">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
:string-one-and-only">
        <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
          AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:physician-speciality"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
```

```
        <Apply FunctionId= urn:oasis:names:tc:xacml:1.0:function
:string-bag>
          <AttributeSelector
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
            Path="//ct:condition[text()=md:record/md:
treatedCondition/text()]/../@ct:name"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
      </Apply>
    </Condition>
  </Rule>
```

P5: The hospital may limit the access to anonymous medical data only to authorized organizations for research purposes.

```
  <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:P5"
Effect="Permit">
    <Description>
      The hospital may limit the access to anonymous medical
data only to authorized organizations for research purposes.
    </Description>
    <Target>
      <AnyOf>
        <AllOf>
```

```
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
            organization
          </AttributeValue>
          <AttributeDesignator
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
            AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:role"
            DataType="http://www.w3.org/2001/XMLSchema#string"
/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:
xpath-node-match">
          <AttributeValue
            DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"
            XPathCategory="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource">
              md:record/md:medical/*[name() != '
personallyIdentifiableInformation']
            </AttributeValue>
```

```
            <AttributeDesignator
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
              AttributeId="urn:oasis:names:tc:xacml:3.0:
content-selector"
              DataType="urn:oasis:names:tc:xacml:3.0:data-type
:xpathExpression"/>
         </Match>
       </AllOf>
     </AnyOf>
     <AnyOf>
       <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
              read
           </AttributeValue>
           <AttributeDesignator
             MustBePresent="false"
             Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:action"
             AttributeId="urn:oasis:names:tc:xacml:1.0:action:
action-id"
             DataType="http://www.w3.org/2001/XMLSchema#string"
/>
         </Match>
       </AllOf>
```

```
      </AnyOf>
    </Target>
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
        <AttributeDesignator
          MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
          AttributeId="urn:oasis:names:tc:xacml:1.0:subject:
subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/
        <Apply FunctionId= urn:oasis:names:tc:xacml:1.0:function
:string-one-and-only>
          <AttributeSelector
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
            Path="//ao:organization/ao:name/text()"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
      </Apply>
    </Condition>
  </Rule>
```

P6: Hospital staff other than physicians may not order lab tests.

```
 <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:P6"
Effect="Permit">
   <Description>
     Hospital staff other than physicians may not order lab
tests.
   </Description>
   <Target>
     <AnyOf>
       <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
             physician
           </AttributeValue>
           <AttributeDesignator
             MustBePresent="false"
             Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
             AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:role"
             DataType="http://www.w3.org/2001/XMLSchema#string"
/>
         </Match>
       </AllOf>
     </AnyOf>
     <AnyOf>
```

```
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:
xpath-node-match">
            <AttributeValue
              DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"
              XPathCategory="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource">
                md:record/md:medical/md:labOrders
              </AttributeValue>
              <AttributeDesignator
                MustBePresent="false"
                Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
                AttributeId="urn:oasis:names:tc:xacml:3.0:
content-selector"
                DataType="urn:oasis:names:tc:xacml:3.0:data-type
:xpathExpression"/>
          </Match>
        </AllOf>
      </AnyOf>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
               order
            </AttributeValue>
```

```
            <AttributeDesignator
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:action"
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:
action-id"
              DataType="http://www.w3.org/2001/XMLSchema#string"
/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  </Rule>
```

P7: A physician may order a lab test except when he is not part of the patient's care team.

```
  <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:P7"
Effect="Permit">
    <Description>
      A physician may order a lab test except when he is not
part of the patients care team.
    </Description>
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
```

```
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
              physician
            </AttributeValue>
            <AttributeDesignator
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
              AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:role"
              DataType="http://www.w3.org/2001/XMLSchema#string"
/>
          </Match>
        </AllOf>
      </AnyOf>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:
xpath-node-match">
            <AttributeValue
              DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"
              XPathCategory="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource">
                md:record/md:medical/md:labOrders
              </AttributeValue>
              <AttributeDesignator
                MustBePresent="false"
```

```
              Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
              AttributeId="urn:oasis:names:tc:xacml:3.0:
content-selector"
              DataType="urn:oasis:names:tc:xacml:3.0:data-type
:xpathExpression"/>
         </Match>
       </AllOf>
     </AnyOf>
     <AnyOf>
       <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
              order
           </AttributeValue>
           <AttributeDesignator
             MustBePresent="false"
             Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:action"
             AttributeId="urn:oasis:names:tc:xacml:1.0:action:
action-id"
             DataType="http://www.w3.org/2001/XMLSchema#string"
/>
         </Match>
       </AllOf>
     </AnyOf>
   </Target>
```

```
   <Condition>
     <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:
any-of">
       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
:string-one-and-only">
       <AttributeDesignator
         MustBePresent="false"
         Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
         AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:physician-id"
         DataType="http://www.w3.org/2001/XMLSchema#string"/>
       </Apply>
       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
:string-bag">
         <AttributeSelector
           MustBePresent="false"
           Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
           Path="md:record/md:careTeam/md:teamMember/md:
registrationID/text()"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>
       </Apply>
     </Apply>
   </Condition>
 </Rule>
```

P8: A person must be a custodial parent or legal guardian must provide proof of kinship to request access to the electronic health record of a patient under the age of 18.

```
 <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:P8"
Effect="Permit">
   <Description>
     A custodial parent or legal guardian may request access to
 the electronic health record of a patient under the age of 18.
   </Description>
   <Target>
     <AnyOf>
       <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:
xpath-node-match">
           <AttributeValue
             DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"
             XPathCategory="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource">
               md:record
           </AttributeValue>
           <AttributeDesignator
             MustBePresent="false"
             Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
             AttributeId="urn:oasis:names:tc:xacml:3.0:
content-selector"
```

```
            DataType="urn:oasis:names:tc:xacml:3.0:data-type:
xpathExpression"/>
         </Match>
       </AllOf>
     </AnyOf>
     <AnyOf>
     <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">
           read
         </AttributeValue>
         <AttributeDesignator
           MustBePresent="false"
           Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:action"
           AttributeId="urn:oasis:names:tc:xacml:1.0:action:
action-id"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>
       </Match>
     </AllOf>
   </AnyOf>
 </Target>
 <Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and
">
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
```

```
       <ApplyFunctionId="urn:oasis:names:tc:xacml:1.0:function:
string-one-and-only">
         <AttributeDesignator
           MustBePresent="false"
           Category="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject"
           AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:parent-guardian-id"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>
       </Apply>
       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
:string-one-and-only">
       <AttributeSelector
         MustBePresent="false"
         Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:resource"
         Path="md:record/md:parentGuardian/md:parentGuardianId/
text()"
         DataType="http://www.w3.org/2001/XMLSchema#string"/>
       </Apply>
     </Apply>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
date-less-or-equal">
       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
:date-one-and-only">
         <AttributeDesignator
           MustBePresent="false"
           Category="urn:oasis:names:tc:xacml:3.0:
attribute-category:environment"
```

```
            AttributeId="urn:oasis:names:tc:xacml:1.0:
    environment:current-date"
            DataType="http://www.w3.org/2001/XMLSchema#date"/>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
    :date-add-yearMonthDuration">
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
    function:date-one-and-only">
            <AttributeSelector
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:3.0:
    attribute-category:resource"
              Path="md:record/md:patient/md:patientDoB/text()"
              DataType="http://www.w3.org/2001/XMLSchema#date"/>
          </Apply>
          <AttributeValue DataType="http://www.w3.org/2001/
    XMLSchema#yearMonthDuration">
            P18Y
          </AttributeValue>
        </Apply>
      </Apply>
    </Apply>
   </Condition>
  </Rule>
</Policy>
```

B.3 Cassandra

P1: Only cardiologists are allowed to access cardiac medical records.

```
permits(x, accessEHR(y, Cardiac)) ← hasActivated(y, Patient),
    hasActivated(x, Physician(Cardiology))
```

P2: No one shall have the ability to delete clinical information until the appropriate time period has expired (6 years).

Cassandra cannot express prohibitions; it assumes everything is prohibited if not otherwise explicitly allowed. We can rephrase P2 such that "clinical information can be deleted after the appropriate time period has expired (6 years)".

```
permits(x, Delete(ehr)) ← CurrTime() - ModificationDate(ehr) >
    Years(6)
```

P3: A doctor should be granted the permissions assigned to the primary care physician of a patient (read and modify his/her EHR) only when the patient has designated him as the primary care physician.

```
canActivate(x, PrimaryCarePhysician(y)) ← hasActivated(pat,Patient
    ()), hasActivated(y, ConsentToPCP(y, x))
permits(x, access(EHR(y))) ← canActivate(x, PrimaryCarePhysician(y
    )), hasActivated(y, Patient), hasActivated(x, Physician)
```

P4: A specialist physician may prescribe drugs if the treated illnesses is related to his/her specialty.

```
Prescribe-drug(pat, drug) ← hasActivated(x, Physician(speciality))
    , Get-patient-info(pat, Illness) \in Get-illnesses-treated-by(
```

```
    speciality)
```

P5: The hospital may limit the access to anonymous medical data only to authorized organizations for research purposes.

```
permits(x, AnonymousEHR()) ← HospitalAdmin.hasActivated(x,
    Authorized(Research))
```

P6: Hospital staff other than physicians may not order lab tests.

Cassandra cannot express prohibitions; it assumes everything is prohibited if not otherwise explicitly allowed. We can rephrase P6 such that "physicians may order lab tests".

```
permits(x, orderLabTests(y, type)) ← hasActivated(y, Patient),
    hasActivated(x, Physician(speciality))
```

P7: A physician may order a lab test except when he is not part of the patient's care team.

```
permits(x, orderLabTests(y, type)) ← hasActivated(y, Patient),
    hasActivated(x, Physician(speciality)), HospitalAdmin.
    hasActivated(x, CareTeam(y))
```

P8: A person must be a custodial parent or legal guardian must provide proof of kinship to request access to the electronic health record of a patient under the age of 18.

```
permits(x, access(EHR(y))) ← Parent(x, y) \vee Guadian(x, y),
    Patient(age) < Years(18)
```

B.4 PeerTrust

P1: Only cardiologists are allowed to access cardiac medical records.

```
hospital:
access(ehr, X) ← cardiologist(X) @ "hospital", cardiac(ehr)
```

P2: No one shall have the ability to delete clinical information until the appropriate time period has expired (6 years).

```
hospital:
delete(X, ehr) \$ X $\leftarrow$ person(X), currentDate() -
   modificationDate(ehr)  >= 6
```

P3: A doctor should be granted the permissions assigned to the primary care physician of a patient (read and modify his/her EHR) only when the patient has designated him as the primary care physician.

```
hospital:
access(ehr, X) \$ X $\leftarrow$ physician(X) @ "hospital" @ X,
   owner(Y, ehr) @ "hospital" @ Y, patient(Y) @ "hospital" @ Y,
   pcp(X, Y) @ "hospital" @ Y
```

P4: A specialist physician may prescribe drugs if the treated illnesses is related to his/her specialty.

```
A specialist physician (e.g., cardiologist) may prescribe drugs if
    the treated illnesses is related to his/her specialty (e.g.,
```

```
    cardiac-related).

hospital:
prescribe(X, Y, drug) \$ X $\leftarrow$ physician(X) @ "hospital"
    @ X, speciality(X, spec) @ "hospital" @ X, patient(Y) @ "
    hospital" @ Y, diagnosis(Y, illness), treatment(illness, drug)
    @ FDA, area(illness, spec)
```

P5: The hospital may limit the access to anonymous medical data only to authorized organizations for research purposes.

```
hospital:
access(ehr, X) ← authorized(X, purpose) @ "hospital", anonymous(
    ehr), purpose = "research"
```

P6: Hospital staff other than physicians may not order lab tests.

```
hospital:
order(X, Y, labTests) \$ X $\leftarrow$ physician(X) @ "hospital"
    @ X, patient(Y) @ "hospital" @ Y
```

P7: A physician may order a lab test except when he is not part of the patient's care team.

```
hospital:
order(X, Y, labTests) \$ X $\leftarrow$ physician(X) @ "hospital"
    @ X, patient(Y) @ "hospital" @ Y, pct(X, Y)
```

P8: A person must be a custodial parent or legal guardian must provide proof of kinship to request access to the electronic health record of a patient under the age of 18.

```
hospital:
access(ehr, X) \$ X $\leftarrow$ owner(Y, ehr) @ "hospital",
   patient(Y) @ "hospital" @ Y, guardian(X, Y) @ stateAuthority @
   X
```

B.5 Ponder

P1: Only cardiologists are allowed to access cardiac medical records.

```
inst auth- CardiologistsCEHR{
  subject <person>  /employee - /physician/cardiologist;
  target  <ehr_details> /cardiacEHR;
  action  read(), write(), update();
}
```

P2: No one shall have the ability to delete clinical information until the appropriate time period has expired (6 years).

```
inst auth- DeleteEHR{
  subject   /person;
  target    /EHR;
  action  delete();
  when ehr.hasExpired = false;
```

```
}
```

P3: A doctor should be granted the permissions assigned to the primary care physician of a patient (read and modify his/her EHR) only when the patient has designated him as the primary care physician.

```
type role PrimaryCarePhysician (PatientEHR patientEHR) {
  inst auth+ PCP{
    subject <person>  s = /physician;
    target      patientEHR;
    action  read(), write(), update();
  }
}
```

P4: A specialist physician may prescribe drugs if the treated illnesses is related to his/her specialty.

Incomplete policy specification—missing "if the treated illnesses is related to his/her specialty" portion of the policy in the Ponder language.

```
inst auth+ Prescribe{
  subject <employee>  s = /physician;
  target  <person>    /patient;
  action  prescribe(drug);
  when  s.speciality = ....
}
```

P5: The hospital may limit the access to anonymous medical data only to authorized organizations for research purposes.

```
inst auth+ Researchers{
  subject <organization>    /university/researcher;
  target  /EHRs;
  action  read();
  {result ehr.anonymized()}
}
```

P6: Hospital staff other than physicians may not order lab tests.

```
inst auth- OrderLabTests{
  subject <person>  /employee - /physician;
  target  <person>  /patient;
  action  order(labTest);
}
```

P7: A physician may order a lab test except when he is not part of the patient's care team.

```
inst auth+ OrderLabTests{
  subject <person>  s = /physician;
  target  <person>  t = /patient;
  action  order(labTest);
  when  s.memberCareTeam(p) = true;
}
```

P8: A person must be a custodial parent or legal guardian must provide proof of kinship to request access to the electronic health record of a patient under the age of 18.

```
inst auth+ ReadEHRGuardian{
  subject s = /person;
  target  t = /patientEHR;
  action  read();
  when t.age <= 18 && (t.parent() = s || t.guardian=s;
}
```

B.6 Rei

P1: Only cardiologists are allowed to access cardiac medical records.

```
has(Physician, right(action(access, EHR), (cardiologist(Physician)
   )))
```

or

```
has(Physician, prohibition(action(access, EHR), (not(cardiologist(
   Physician)))))
```

P2: No one shall have the ability to delete clinical information until the appropriate time period has expired (6 years).

```
has(Person, prohibition(action(delete, EHR, (notExpired(EHR)))))
```

P3: A doctor should be granted the permissions assigned to the primary care physician of a patient (read and modify his/her EHR) only when the patient has designated him as the primary care physician.

Cannot express the target as a function that returns a specific EHR (EHR of Y).

```
has(Physician, right(action(read, [EHR of Patient], (
   primaryCarePhysician(Physician, Patient)))))
```

P4: A specialist physician may prescribe drugs if the treated illnesses is related to his/her specialty.

```
has(Physician, right(action(prescribeDrug, Patient, (
   patientIllness(Patient, Illness), prescribedDrug(Patient, Drug)
   , treatment(Drug, Illness), speciality(Physician, Speciality),
   area(Illness, Area), Speciality = Area))))
```

P5: The hospital may limit the access to anonymous medical data only to authorized organizations for research purposes.

```
has(Organization, right(action(read, anonymousEHR), purpose(
   Organization, research)))
```

P6: Hospital staff other than physicians may not order lab tests.

```
has(Staff, prohibition(orderLabTests, not(Staff, physician)))
```

P7: A physician may order a lab test except when he is not part of the patient's care team.

```
has(Physician, right(action(orderLabTests, Patient, inCareTeam(
   Physician, Patient))))
```

or

```
has(Variable, prohibition(action(orderLabTests, Patient, not(
    inCareTeam(X, Y)))))
```

P8: A person must be a custodial parent or legal guardian must provide proof of kinship to request access to the electronic health record of a patient under the age of 18.

```
has(Person, right(action(read, [EHR of Patient, age(Patient, N), N
    < 18)), presentsProofOfKinship(Person, Patient)))
```

Appendix C

PrimaCare's data model

- Health record management
  - Patient demographics
  - Patient contact information
  - Allergies
  - Family history and relationships
  - Medical and social history
  - Vital signs
  - Medications
  - Encounters (complaints or reason for encounter)
  - Consultation summary
  - Diagnosis and desease notification
  - Treatment
  - Interventions or procedures
  - Immunizations

  - Laboratory orders and results
  - Doctor's remarks
  - Counseling
  - Imaging order and result
  - Prescriptions
  - Referrals
- Standardized concepts
  - Symptom
  - Laboratory and clinical observations
  - Diagnostic
  - Diagnose
  - Deseases
  - Treatment
  - Drugs
- Administrative
  - Accounting
  - Billing (medical, laboratory, imaging)
  - Insurance
  - Room booking
  - Provider's assets
  - Clinic information and departments
  - Medical supply management

- Supplier management
- Referral centers
- Staff management
- System management