Knowledge-based Cyberinfrastructures for Decision Making in Real-World Domains

Thesis
Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By
Shubhanan Veerendra Deshpande, B.E.

Graduate Program in Computer Science and Engineering

The Ohio State University
2011

Thesis Committee:

Dr. Jay Ramanathan (Advisor)
Dr. Rajiv Ramnath
Dr. Ola Ahlqvist
Copyright by
Shubhanan Deshpande
2011
Abstract

Contemporary domain software such as gaming, training and simulation software successfully captures objects and interactions in the corresponding real-world domains to enable learning and decision-making. However, such systems are designed, developed and operated in isolation, despite the fact that they have objects and interactions that are strikingly similar in the real world. Their features do not exploit commonality, and hence, are a result of independent characterization of a subset of underlying reality. Such disjoint domain modeling has lead to specialized, but isolated software features that ought to be connected as the domains in real world intrinsically are.

In this thesis, we propose a cyberinfrastructure framework that allows domain objects, interactions and decisions, and derived knowledge to be inter-related using shared upper ontology. This allows multiple simulation, gaming, and collaboration features to be composed and utilized in new ways to lead to decisions and learning that is more consistent with the real-world. The research enables specialized, dispersed communities of practice to easily leverage knowledge from other related real-world collaborations and solutions, while they continue to collaborate in their own domains of interest. This thesis motivates and develops a framework that is used for connecting real-world domain software at multiple levels – from the domain model to implementation constructs. It implements a reference architecture prototype and applications that illustrate making the tacit knowledge explicit for sharing and reusability. It also sets directions and some leads for the future research.
Dedication

To my family
Acknowledgements

I would like to thank Dr. Jay Ramanathan for her motivation, support, guidance and patience throughout the work in this thesis. Thanks to Dr. Rajiv Ramnath, for his help and enthusiasm, particularly during implementation phases of this work. Their continued support and assistance, thesis related and otherwise are highly appreciated.

I would also like to thank Dr. Ola Ahlqvist for his valuable contribution in the motivation for this thesis. The discussions with him in late spring 2009 and his work in teaching Green Revolution through simulation have been solid subjects of study and motivation for this thesis.
Vita

2002 to 2006......................... Bachelor of Engineering (B.E.) in Computer Engineering, College Of Engineering, Pune

Pune University, India

2005 to 2006......................... Tata Research Development & Design Centre, Pune, India

2006 to 2008......................... Yahoo! R&D, Yahoo! SDC, Bangalore, India

Jan2010 to Dec2010............... Graduate Research Associate, Memory & Language Processing Lab, and CERCS for Enterprise Transformation & Innovation Lab, Dept. of Computer Science & Engineering, The Ohio State University, Columbus, Ohio, USA

Fields of Study

Major Field: Computer Science and Engineering
# Table of Contents

Abstract .............................................................................................................................. ii
Dedication .......................................................................................................................... iii
Acknowledgements ........................................................................................................ iv
Vita ................................................................................................................................. v
List of Tables .................................................................................................................. x
List of Figures ................................................................................................................ xi

1. Introduction .................................................................................................................. 1
   1.1 Problem Context ........................................................................................................ 1
   1.2 Research Impetus ....................................................................................................... 3
   1.3 Background Research ............................................................................................... 4
   1.4 Research Contributions ............................................................................................ 8
   1.5 Thesis Outline .......................................................................................................... 9

2. Problem Formalization and Validation Dimensions .................................................... 11
   2.1 Terms & Definitions ................................................................................................ 11
   2.2 Problem Statement ................................................................................................. 13
   2.3 Validation Dimensions ........................................................................................... 14
3. Illustrative Case Studies of Scapes

3.1 SRMN Collaboration: A STEM Education Simulation

3.2 HealthGame: A Human Well-Being Simulation

3.3 GreenCity: An Energy Management Simulation

3.4 RealtyHub: A Realty Search & Pricing Application

3.5 The (Lost) Real-World Interconnection: The Operational Validation

3.6 Sufficiency as Validation Examples

4. The Cyberinfrastructure Framework: Mirror

4.1 Introduction to the Mirror Framework

4.2 Architecture Approach

4.3 Mirror Ontology Specifications & Mappings

4.4 Shared Upper Ontology for Scape/Core Dynamics

4.4.1 Design Validation: Completeness of Shared Upper Ontology

4.5 Framework Reference Architecture and Design

4.5.1 Key Design Considerations & Solutions

4.5.2 Design Validation: Qualities of Framework Design

4.6 Implemented Reference Architecture: Mirror Core

4.6.1 Ontology Specifications in the Core

4.6.2 Implemented Core Services

4.7 Scape Configuration & Development Process
5. Implemented Reference Architecture: Scapes................................................................. 103

5.1 HealthGame Scape.................................................................................................. 103

5.2 GreenCity Scape.................................................................................................... 129

5.3 RealtyHub Scape .................................................................................................. 131

5.4 Framework Operational Validations ....................................................................... 132

6. Summary of Validations .......................................................................................... 137

6.1 Framework Design Validations .............................................................................. 137

6.1.1 Completeness of the Shared Upper Ontology....................................................... 137

6.1.2 Applicability of the Framework .......................................................................... 137

6.1.3 Sharing & Re-Use .............................................................................................. 137

6.1.4 Evolution & Road Map ....................................................................................... 138

6.1.5 Framework Qualities ......................................................................................... 138

6.2 Framework Operational Validations ....................................................................... 138

6.2.1 Establishing a Shared Understanding of the Underlying Reality......................... 138

6.2.2 Scape Configuration & Development Process .................................................... 138

6.2.3 Understanding and Interconnecting Domain Objects & Behavior ..................... 138

6.2.4 Demonstrating Sharing & Re-Use ..................................................................... 139

7. Conclusion .................................................................................................................. 140

8. Future Work ................................................................................................................ 142

References ...................................................................................................................... 143
Appendix A: SRMN Solution Design & Implementation Details................................. 148
Appendix B: Mirror Core Implementation Details...................................................... 155
Appendix C: A study of Cyberinfrastructures.............................................................. 174
List of Tables

Table 1: Terms and Definitions ........................................................................................................... 11
Table 2: A scenario in HealthGame ..................................................................................................... 18
Table 3: Mirror Prototype Governance Structure ............................................................................... 164
Table 4: Core Schema Description ..................................................................................................... 169
Table 5: Example (existing) Cyberinfrastructures ............................................................................. 176
List of Figures

Figure 1: A sample scenario execution in a domain software............................19
Figure 2: The Mirror Framework Operational Snapshot ....................................28
Figure 3: Representation & Inter-Relation of Scape Dynamics in Mirror ..................34
Figure 4: Sample Mirror Upper Ontology for Domains......................................35
Figure 5: Examples of Domain Mapping for Scape Objects..................................37
Figure 6: Shared upper ontology Snapshot ..........................................................43
Figure 7: Domain Mapping for Second Life on Mirror .........................................47
Figure 8: Domain Mapping for GeoGame on Mirror .............................................48
Figure 9: Domain Mapping for HealthGame on Mirror .........................................49
Figure 10: Mirror Framework Reference Architecture .........................................51
Figure 11: Mirror M-V-C and Technology Stack ....................................................53
Figure 12: Mirror Core Component Architecture ..................................................54
Figure 13: Building Mirror Shared Upper Dynamics Ontology ............................57
Figure 14: Implementation for Import/Export Services in the Core .......................89
Figure 15: Implemented Import/Export Services in Mirror Core ..........................90
Figure 16: A Data Sharing and Mediation Service Implementation ........................91
Figure 17: A Data Sharing & Mediation Service Definition ...................................92
Figure 18: Sharing and Re-Use in Mirror Framework ..........................................93
Figure 19: Implemented Core Objects & Services at the backend (Model in MVC) ....94
Figure 20: The front end (View & Controller in MVC) for Core Entity Services .......95
Figure 21: Administrative UI (View in MVC) to the Core Services .......................96
Figure 22: Runtime Services: Logging Capabilities in Mirror Core .......................97
Figure 23: Runtime Services: Rule Engine in Mirror Core ..................................98
Figure 24: Scape Development Process ..............................................................100
Figure 25: Scape Development Semi-Automation ...............................................102
Figure 26: HealthGame Object Interaction/Domain Model ........................................ 104
Figure 27: HealthGame Scape Implementation ....................................................... 129
Figure 28: GreenCity Domain Ontology ................................................................. 130
Figure 29: GreenCity Scape Implementation ......................................................... 130
Figure 30: RealtyHub Domain Ontology ............................................................... 131
Figure 31: RealtyHub Scape Implementation ......................................................... 131
Figure 32: Rule Execution with Shared Upper Ontology in the Core ...................... 132
Figure 33: Understanding and Expressing Scape Objects & Behavior .................... 133
Figure 34: Making the Assumptions Explicit ....................................................... 134
Figure 35: Enabling the Domain Interconnection .................................................. 135
Figure 36: Enabling the Domain Interconnection .................................................. 135
Figure 37: SRMN Enterprise Model ...................................................................... 149
Figure 38: SRMN Governance ............................................................................. 150
Figure 39: SRMN Roles and Workflow .................................................................. 153
Figure 40: SRMN Solution Implementation .......................................................... 154
Figure 41: Core User Stories - Prioritization and Iteration Management ............... 165
Figure 42: Mirror Core Sample Use-Cases-1 ....................................................... 166
Figure 43: Mirror Core Sample Use-Cases-2 ....................................................... 166
Figure 44: Mirror Core Sample Use-Cases-3 ....................................................... 167
Figure 45: Shared Ontology used to derive the Core Schema ................................ 168
Figure 46: Mirror Core Operation ........................................................................ 172
Figure 47: Core Flex (View + Controller) Code Structure .................................... 172
Figure 48: Core EJB (Model) Code Structure ...................................................... 173
Figure 49: Cyberinfrastructure-Technology De-coupling ..................................... 182
1. Introduction

1.1 Problem Context

Specializing software for a set of objects and interactions in a constrained subset of real world (i.e. a domain) was greatly helped by domain modeling methodologies, to enable learning and decision-making within the domain. Some examples of such software are farming simulations, health care record management, finance (insurance, banking) applications, urban planning simulations, etc. The increasing domain focus naturally required modeling assumptions and ignorance about factors external to the domain. But on the contrary, the rationality of decisions and learning ability provided by any software is bounded by the information available to that software. So, any simulation, training or gaming gets inconsistent with the real world when such ignored and assumed factors affect the domain elements indirectly, and are subject to change dynamically. Decisions become irrational in the real world because of lack of information available to arrive at the decisions in domain software. Such situations are a natural result of the appreciation and pervasiveness of open source methodology, which encourages dispersed ‘communities of practice’ to engage in enhancing their specific domains of interest. The disjoint domain modeling has therefore lead to isolated stacks or silos of software capabilities, which ought to be connected like the real-world domains. For example, an architecture simulation for a city can assume little about the traffic pattern in that city in order to do any urban planning for the city; insurance costs have to know about a person’s medical records, credit history and driving record; an energy management simulation will affect the supply-chain, ecology and health around the supplying power station, and so on.

* Community of Practice: A group of people with shared interest, skill or profession [41]
Now, not only do the communities of practice need to communicate within and across, but they also need to put their work together. To support such advanced collaboration, we need to co-relate the domains of interest and the work-products together, and enable sharing and re-use of knowledge or other resources.

Consequently, the next generation software has to cope with a great deal of knowledge exchange and synthesis, while simultaneously maintaining traceability amongst interaction work-products (user decisions and knowledge), across the geographical space and time [4]. Software applications to enable some such aspects, and help decision making ability do exist. For example, Facebook for networking and communication, knowledge based AI systems, virtual simulations and games like Second Life, GeoGame and other strategy games, Record Management Systems like Customer Record Management or Patient Record Management systems, and so on. However, comprehensively supporting real-world human (inter)actions and understanding the behavioral patterns consistent with real-world constraints cannot be addressed by such disjoint partial solutions.

The fact that similar real-world objects and interactions are modeled in multiple domains can be exploited for the inter-connectivity; but the domain software is totally oblivious of this fundamental commonality in real-world domains. For example, an agent modeled in war strategy games, is similar to a farmer modeled in a farming simulation, because both games are actually talking about the same entity - “person” in real world. But neither the specifics of “agent” or “farmer” nor the underlying idea of “person” is known to such family of software. Since such objects can’t be related, neither their domain interactions and nor the derived intelligence can be related together towards decision-making or learning in the real world.

So, the main challenges are to express and interpret the common static and dynamic aspects of individual components as a shared upper ontology† model, and to combine the knowledge derived from the individual domain software. With this aim, we are forced to shift gears from just developing software, to also include efforts oriented towards a more systemic and

† Ontology: A formal representation of knowledge and behavior as a set of concepts within a domain, and the relationships between those concepts
integrative approach for making distributed software applications work together on an infrastructure.

In this thesis, we show how objects and behavior in real-world domains could be modeled to enable traceability\(^\ddagger\) between decisions across domains, allowing the inherently different domain software to function together with shared knowledge.

### 1.2 Research Impetus

Any assessment, inter-connection and knowledge mining is not possible unless software(s) share information across themselves. We need to incorporate the real-world inter-connections into software domains and components to enable usable decision-making and learning ability in a more complete context in the real world, i.e. exposed to intrinsic and external real-world constraints. Precisely, we -

- Need to co-relate the domains, and the narrowed models of reality
- Need to mediate differences and heterogeneity
- Need a shared upper ontology
- Need to support component re-usability

This research enables decision traceability and combination, and benefits in scenarios where there is a need to leverage knowledge from other real-world collaborations or solutions. For example,

- **Government Organizations**

  Typical government organizations need to make sense of operational data and knowledge from multiple agents, like organizations, processes and their participants, in order to make policies that lead to more effective governance [46, 47]. Often, this

\(^\ddagger\) Traceability: is the ability to trace or track entities based on some relation
requires dynamically combining decisions from different active agents in complex, chaotic environments. An example scenario in emergency management, and the need for dynamically combining related decisions, is identified and emphasized in [43]. As another example, crucial performance enhancement insights were achieved when the ODJFS\(^5\) organization operational/performance data could be related to that of another organization, City of Columbus, which was performing better [46]. This inter-relation was possible only through the use of shared upper ontology, presented in Chapter 4 and 5, and visualization tools for the mined data.

- Simulations, Learning & Strategy games

Such software can focus on the domain interactions, but get away with the incorrectness (owing to assumptions or ignorance about external factors) in their imparted learning, or derived decisions. For example, crash test simulations, distributed learning agents like robotic arms can learn from each other, ERP\(^*\) systems can interconnect decisions from its different modules, etc.

The following sections introduces the reader to the relevant background research towards the set of problems and goals identified till section 1.2; and explains how we further it to address the problem in the context of this thesis.

### 1.3 Background Research

The research motivation that we took from the problem is mentioned in section 1.2. Following section provides the important background research related to the motivation, that this thesis builds on. This is categorized into five major categories: Ontology & Knowledge Expression, Software Engineering & Product lines, Software Evolution, Enterprise Architecture and Application Integration, and Cyberinfrastructures. The

---

\(^5\) ODJFS: Ohio Department of Job and Family Services

\(^*\) ERP: Enterprise Resource Planning
derivations from this background research are further discussed in chapter 4, when we introduce and explain the proposed framework.

**Ontology and Knowledge Expression:** Though OWL†† has a lot of expressive power and inference ability, it lacks the power of a programming language for taking descriptions to programmatic implementations. [1] talks about bridging the gap between OWL and other static and dynamic typed programming languages. Various combination approaches OWL and Java are discussed in [2] and [8]. A short summary of design patterns with Java Enterprise Edition is presented in [3]. An approach to rapidly integrate heterogeneous data sources and aggregate events by establishing domain ontology is presented in [5]. Work in [9] presents a mechanism for evolving a database schema with dynamic requirements of object-oriented software. [11] presents methods to integrate and relate specific ontologies‡‡ to generic upper ontologies like Cyc; whereas extending an upper ontology like SWEET to make it more specific to a domain is described in [12]. [13] describes usage and enhancing approaches for upper ontologies. [34] explains an example upper level taxonomy from which an upper level ontology can be derived.

**Enterprise Architecture and Application Integration:** This research aims at enabling evolution of domain software together to simulate interactions in a consistent manner with the real world, and ultimately map the entire real world into software. Towards this, we realized the importance of viewing the enabled “mirrored”§§ world as the enterprise driving the proposed framework. This helped discover that some of the problems we saw as imminent while designing the framework, are typical of enterprise applications. [14], for example, describes integrating enterprise value chains using layered ontology approach, while ontology-based enterprise architecture approach, for enterprise integration to address dynamically changing requirements, is described in [15]. Applying laws and conventions to

---

†† Web Ontology Language

‡‡ A list of sample ontologies can be found on the Stanford’s “Protégé Ontologies” website (http://protege.stanford.edu/download/ontologies.html)

§§ The key difference between Virtual world and Mirrored world is that the later is tied to reality while former lies close to fiction. The name for the proposed framework is derived from [57]
agents based on their context of operation is discussed in [17]. [27] describes a framework for combining, using knowledge sharing, several agents in a multi-agent system working to solve a complex problem. [19] discusses design and implementation to enhance transparent synchronous collaborative tools to let different collaboration groups work together. Approaches such as Enterprise Application Integration work mainly on data integration and transformation, and are not suitable for dynamic environments owing to their maintenance cost over time & scale, and low flexibility [26]. [45] describes notation and semantics for goal modeling, and linking the organizations’ dynamic operations to its goals, using the ACE framework and RED transactions as described in [30].

**Software Engineering and Software Product Line Architecture:** [44] describes a decision-making framework for translating software requirements in a multi-stakeholder environment into architectural evaluations, while [29] sets an architectural decision framework for complex systems to reflect economical and technical issues in the architectural design. [33] explores a goal-driven collaboration framework for serious gaming, and demonstrates the usage for a farming application, while [32] explains a sense-respond framework for adaptive participatory services. The scenarios built in [32] and [33] are used as few of the case studies to evaluate this research. [48] describes a way to combines SOA and software product line principles; it presents a combined approach in which SOA applications are developed as software product lines to achieve high customizations and planned reuse. [49] provides a descriptive report on a framework for software product line practice. [51] addresses the specification and representation issues related to software product line architectures.[50] motivates the idea that software product lines, domain engineering, reuse and reengineering are intimately related with each other.

**Software Evolution:** [52] analyzes empirical data on commercial software to test and better understand software evolution, and to measure effects of software automations on the software productivity and quality. [53] proposes a classification of software evolution and maintenance processes actually carried out over the years in industry; this helps further understanding the software evolution and maintenance processes. [54] describes the

*** Service-Oriented Architectures: approach for developing distributed applications as a set of self-contained, business-aligned services. [48]
phenomenon of software evolution and its impacts as the domains co-evolve with the software. [55] discusses the software evolution of two real world software case studies, which have employed product line architecture approach for several years. It concludes with the specific lessons learnt and general guidelines for product line architectures in evolution. [56] presents basic mechanisms to support software product line process evolution.

**Cyberinfrastructure:** The scope of this problem made it indispensable to study and exploit “cyberinfrastructure” features while designing the proposed framework. NSF coined the term “cyberinfrastructure” through its Blue-Ribbon advisory panel, to refer to any large scale, distributed software system that connects people, technologies, computers and data to support derivation of novel scientific theories and knowledge [42]. Among other good descriptions of cyberinfrastructures is one expressed in [21], which goes by - cyberinfrastructure refers to the integration, coordination, and deployment of information technology and human resources to support modern science and engineering problems.

The internet remains as a grand but ad hoc collection of isolated sub-infrastructures. Advanced knowledge infrastructure, by contrast, is human-centered and optimized for particular resources and communities. A cyberinfrastructure enables a vision of advanced knowledge and collaboration infrastructure that integrates diverse resources across geographical and time barriers, and across discipline, community, sector, and jurisdiction. Bearing a human-centric approach to control diverse resources, cyberinfrastructure must also be sensitive to institutional, legal, and cultural context apart from the technological aspects. Allied research attempts formalizing the cyberinfrastructure concept and acknowledges it as being a nascent revolution in the current industry. [6], [20], [22], [26], [37], [38], [39] and [40] together provide a complete overview of the concept of cyberinfrastructures – including typical cyberinfrastructure characteristics and operational environment, advanced collaborations in a cyberinfrastructure, cyberinfrastructure process and design, future steps and the issues and challenges in designing and implementing a cyberinfrastructure. [21] describes the processes to develop an integral and integrated set of cyberinfrastructure programs that can support research and education efforts, and relay to a changing environment.
1.4 Research Contributions

In this section, we detail the contributions to the existing research through this thesis.

Efforts identified in section 1.3 greatly helps in establishing semantic understanding between agents or software, but only in specific set of directly related domains, on a point-by-point basis only at an application scale within an enterprise. There is, however, no easy way to semantically interconnect the dynamics of any distributed simulation, training, gaming or other domain software with another, possibly across enterprises with differing business goals, to get away with implicitly modeled assumptions and ignorance about (seemingly) external factors. Consequently, there is no way to dynamically combine the user decisions and derived knowledge from any and multiple domain software operating in distributed environments, for a decision-making or learning ability consistent with real-world constraints.

Through this research, we provide a reference architecture for a cyberinfrastructure, that models a widely applicable shared upper ontology to establish inter-connect between any disparate domains and their enabling software, and make explicit the otherwise ignored tacit knowledge and implicit assumptions.

The contributions of this thesis are in terms of:

A solution framework:

- Prototype Use Cases
- Features & Design Considerations
- Components & Reference Architecture
- Use Cases for Evolution

Implementations:

- Mirror Core (the Reference Architecture)
• Process of implementing domain applications with the framework

Validations of the Framework:

Design Validations:

• Completeness, Applicability, Sharing & Re-Use, Evolution of the framework
• Quality of Framework Design†††

Operational Validations:

• Establishing a shared understanding in the example domains
• Understanding the domain software objects and behavior
• Inter-Relating the domains, domain objects, dynamics and derived knowledge
• Re-using & sharing of components like objects, services, processes, knowledge, and so on.

1.5 Thesis Outline

Chapter 2 formalizes the problem into a formal statement, and provides validation dimensions for the proposed framework. Chapter 3 presents different software applications as case studies, to help demonstrate the problem and emphasize its impact. Chapter 4

††† measured using relevant typical software quality factors like extensibility, scalability, flexibility, Interoperability, Security, and Consistency in the context of this work. Other factors such as Availability, Efficiency, etc are not prime concerns for the prototype implementation of this framework. Scalability is currently not considered because its proposition and evaluation requires extraneous components like caches, load balancers, which are currently out of scope for demonstrating the proof of concept. However, availability and scalability are easily incorporable in the framework owing to the design qualities described in section 4.5.2.
provides the details about the architecture, design and operation of the framework proposed to establish a shared understanding and semantic inter-connect between distributed domain software. It also provides the implementation of the framework reference architecture, shared upper ontology. Chapter 5 presents the re-implementations of case studies discussed in Chapter 3 using the framework. Benefits and usages of the framework are presented through Chapter 4 and 5; and the research validations are summarized in Chapter 6, where the framework is evaluated against the problem statement and the validation dimensions. Chapter 7 summarizes the concluding thoughts on the work through this thesis, and Chapter 8 identifies directions for furthering the proposed framework. Appendices detail the implementations referred in Chapters 4 through 7.
2. Problem Formalization and Validation Dimensions

This chapter defines the terminology used extensively through this thesis, and then formalizes the problem, and re-states the validations using the terminology.

2.1 Terms & Definitions

<table>
<thead>
<tr>
<th>“Domain” or “Context”</th>
<th>A subset of real-world that has objects and (inter)actions in real world. E.g. Farming, Healthy Living, Patient Record Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Scape”</td>
<td>Software that simulates objects and (inter)actions in a domain. E.g. GeoGame, HealthGame, CRMs</td>
</tr>
<tr>
<td>(software) “Framework”</td>
<td>An abstraction with reusable, common code with generic functionality, available for selective overriding/specializing. A framework typically also includes guidelines, procedures and workflow to be able to extend the generic functionality to a specific application.</td>
</tr>
<tr>
<td>“Reference Architecture”</td>
<td>An architectural template solution, with common vocabulary for implementations. (Helps stress the commonality)</td>
</tr>
</tbody>
</table>

Table 1: Terms and Definitions
<table>
<thead>
<tr>
<th>“Mirror Core”</th>
<th>Software that expresses, shares the underlying commonality in different Scapes, enables the “understanding” &amp; “implications”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Cyberinfrastructure”</td>
<td>Technological solution to efficiently connect data, computers, and people with the goal of enabling derivation of novel scientific theories and knowledge.</td>
</tr>
<tr>
<td>“Mirror” Framework</td>
<td>Cyberinfrastructure framework solution for evolving the Scapes and core together</td>
</tr>
<tr>
<td>“View”</td>
<td>The objects (and associations) of reality that the Scape models</td>
</tr>
<tr>
<td>“Service”</td>
<td>A functionality provided by Scapes/Core</td>
</tr>
<tr>
<td>“Ontology”</td>
<td>A formal representation of knowledge and behavior as a set of concepts within a domain, and the relationships between those concepts</td>
</tr>
<tr>
<td>“Evolution”</td>
<td>Expansion of the whole system (Core+Scapes) to model more contexts</td>
</tr>
</tbody>
</table>

Table 1 Continued
Table 1 continued

<table>
<thead>
<tr>
<th>“Ontology”</th>
<th>A formal representation of knowledge and behavior as a set of concepts within a domain, and the relationships between those concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Evolution”</td>
<td>Expansion of the whole system (Core+Scapes) to model more contexts</td>
</tr>
<tr>
<td>“Traceability”</td>
<td>The ability to trace or track entities based on some relation between them</td>
</tr>
</tbody>
</table>

2.2 Problem Statement

Using the above set of terminologies, the formal problem statement for this research work is:

*Create a cyberinfrastructure framework, that*

- *Shares knowledge across multiple communities of practice, and supports human (inter)actions in real world,*

- *Supports evolution by*
  - *maintaining and linking multiple Scapes*
  - *expressing knowledge and behavior*
  - *supporting sharing, re-use and configurable services*
2.3 Validation Dimensions

The proposed framework is validated using following dimensions:

Design Validations:

- **Completeness**: of expression of the shared upper ontology representing the commonality amongst real-world Scapes. This dimension has two aspects, based on how widely the shared upper ontology is applicable to model the following, in order to declaratively and effectively describe the Scapes -
  
  - Real-world Objects, and
  
  - Real-world Dynamics subject to the rules or constraints

- **Sharing & Re-Use**: to express the Scape based on the shared upper ontology, and enable reuse in terms of objects, functionality (services) and knowledge.

- **Applicability**: An implementation of the framework with Mirror Core and example Scapes is realizable. We describe the example Scapes and their sufficiency for evaluation in Chapter 3.

- **Evolution**: the ability of the framework to embrace change\(^\dagger\). The evolution ability is argued at this time since only a prototype implementation of the framework and implementations of only few Scapes are available. This argument is based on background research in software evolution, use of technologies while maintaining flexibility and dynamic configuration, and so on, as described in Chapter 7.

- **Quality of Framework Design**: This includes a study of the framework design against ideally expected software qualities. These are measured using typical Software Engineering metrics like extensibility, interoperability, flexibility, and security.

\(^\dagger\) Change in the operational environment of the framework, i.e. framework requirements and in component implementations.
Operational Validations:

- Establishing a Shared Understanding through Mirror Core
- Process of implementing Scapes with the framework
- Understanding Scape Objects & Behavior
- Inter-Relating Scape Domains, Objects, Dynamics and Derived Knowledge
- Re-Using & Sharing of Framework components like objects, services, processes, knowledge, and so on.

These validations are presented in Chapters 4 to 5, and summarized in Chapter 6 when the framework concept and its reference architecture implementations have all been introduced.
3. Illustrative Case Studies of Scapes

This chapter introduces examples of Scapes, and identifies the shortcomings that directly or indirectly hamper their decision-making or learning ability. These examples span from real-world applications to research prototypes built in labs. The first few sections (3.1-3.4) introduce the Scape case studies; and later, section 3.5 demonstrates the implicit assumptions, ignorance and the missing semantic inter-connects between them. The last section provides the sufficiency of these cases for validation purposes.

3.1 SRMN§§§ Collaboration: A STEM Education Simulation

SRMN project dealt with a collaboration of visionaries from the fields of education, research and the industry. Details of the enterprise organization, governance framework and solution implementation for SRMN project are detailed in Appendix A.

The ultimate goal is to increase the participation of students in STEM careers by a factor of two till the year 2015. Raytheon Inc. took an initial attempt at understanding the factors and student decisions that drive the students towards or away from participation in STEM careers; and represented such dynamics formally as a simulation model (hereafter referred to as the ‘STEM model’) using System Dynamics Modeling concepts.

§§§ STEM Research & Modeling Network: [http://stemnetwork.org], STEM refers to Science, Technology, Engineering & Mathematics
The model thus developed did not serve as a robust pipeline through which captured data could be processed to get outputs to derive policies; however, it certainly was a template that could be tweaked according to individual school and locality scenarios in order to construe the current situation and develop strategies and policies for future. The STEM model was still seen to be short of other contributing factors and variables such as socio-economic situations and decisions, student family background, health conditions, etc and was decided to put forth to open electronic discussion, verification and contribution from expertise around the world. A software solution for collaboration and configuration management was implemented with the aim to build a workbench for the dispersed researchers, educationalists and industrialists to:

- convene electronically to collaboratively improve the STEM model
- develop and discuss the correct policies as the outputs of this effort
- combine and put forth these policies in front of the administrative government to increase the students’ participation in STEM careers.

More details on this solution implementation are provided in Appendix A.

### 3.2 HealthGame: A Human Well-Being Simulation

HealthGame is interactive gaming software that senses the users’ current status using various sensing agents, and proposes the users next possible actions of interest towards the goal of living healthy. The propositions are made based on the sensed environment and the past user decisions, through a sense-respond framework presented in [32].

Problem situations are presented to the user to capture the context and decision making knowledge in a particular scenario. This captured knowledge is then used to shape the users’ future decisions. The goal of the scenario presented in the game is to make the users aware of alcohol consumption risks. The table below shows a “driving drunk” scenario as specified and implemented in [32].
Driving Drunk:

- The player has entered in the amount of alcohol they have consumed and the time frame
- The game calculates their blood alcohol content and determines that the individual is over the legal limit of .08%
- This initiates a display of options and they spin a wheel to be dealt a result of either:
  - Drive home
  - Walk home
  - Call a cab/friend
- Hypothetically speaking: the player receives the option of driving home and they spin the wheel again to determine what happens to them. They will either make it home safely, get in an accident, or get pulled over and arrested
- If they make it home safely: a tutorial explaining the risks associated with their decision would pop up for educational purposes
- If they get in an accident: the severity would be another randomized factor that could range from minimal damage/injury to death or death of others
- If they get pulled over and arrested: they will have to deal with the consequences and use the game to locate resources on how to handle them.

Table 2: A scenario in HealthGame

The following figure helps explain interactions between user and the game that lead to some final state or goal depending on the user decisions through the interaction process.

**** Table borrowed from [32] for explaining the scenario as implemented in [32]. This thesis uses the applications developed in [32] and [33], along with its own implementations, for research validation purposes.
More details on the design and implementation of HealthGame are provided in [32].

3.3 GreenCity: An Energy Management Simulation

The GreenCity is a GIS based energy management simulation that enables its users to optimize the energy usage in a city. It allows the users to manipulate various parameters like energy sinks at different geographic locations in a city, and thus simulate the increased requirement for energy in a city. This requirement can then be alleviated by placing transmission boosters, transformers at different locations to optimize the transmission losses...

Figure 1: A sample scenario execution in a domain software††††

†††† Figure borrowed from [32] to explain the scenario execution, as implemented in [32]. This thesis uses the applications developed in [32] and [33], along with its own implementations, for research validation purposes.
within the city and its outskirts. A user can also deploy energy-conversion equipments like windmills at different locations to help towards the increased energy requirement for the productivity of the city. All these user decisions are incorporated to calculate the simulated energy usage for a city. Ultimately, the user is at a state where he/she can propose an energy optimization plan to the government or any other relevant administration, based on the modeled energy usage and various transmission or local energy-production equipment deployments at different geographic locations.

3.4 RealtyHub: A Realty Search & Pricing Application

RealtyHub is a transaction based software application that allows the user to view the realties and other facilities like office, gyms, and pools available in an area. It shows the user all the statistics like base rate, taxes, miscellaneous fees, contact information, and so on for a particular property.

When the user decides to purchase any property, it accepts the request and on the completion of transaction, updates its database to reflect the latest transaction.

3.5 The (Lost) Real-World Interconnection: The Operational Validation

This section re-emphasizes the need for the shared upper ontology and specific domain ontology within the framework, using the examples described in sections 3.1 to 3.4. It identifies the problems with the decisions and learning derived from the domain software; and also demonstrates how these seemingly disparate games and simulations should actually be interconnected, simply because their real-world domains are. The lack of this interconnect compromises the rationality of decisions and learning ability provided by these domain software.

The problems and the inherent disconnect, identified in this section, also serve to the operational validations for the research presented in this thesis. Further discussion on why
the examples mentioned in sections 3.1-3.4, are sufficient for such validation is presented in section 3.6.

Following paragraphs identify the problems in the domain-specific implementations, and the examples of impact of the inter-domain disconnects inherently modeled in these existing domain applications.

- The SRMN Collaboration for STEM Education Simulation:

  The multiple communities of practice quickly evolved their own version with domain specific simulation capabilities. This lead to multiple versions and induced the ‘boiler-plate’ and ‘silo’ effect in the STEM model development, thereby making it more and more difficult to utilize the STEM model as a strategic asset for integrative decision making. Precisely, the major issues faced were:

  1. The idea of ‘change’ identified by the dispersed communities of practice – not only in the external requirements, but also in the composition of services that serve them, needed to be embraced in the solution; i.e. the idea of micro-change needed to be propagated through various versions of the original model.

  2. Data inputs were heterogeneous, from different sources. As the model by itself was a template for one type of education subsystem, it had to be tweaked along different dimensions for each and every school or state that had rather different set of dynamics than those modeled in the STEM model. Mediation of data heterogeneity and semantic differences emerged as a critical need.

  3. The audience of this enterprise included the users that have the expertise to play with the model (e.g. Researchers, Educationalist, System Dynamics Engineers, etc) as well as users who did not have any knowledge about the model representation and simulation (E.g. Social workers, government, etc). The latter sets of users were rather interested in extracting knowledge from the generated data. Clearly, one data set had to be mined in different ways to provide different information to different viewers and stakeholders (e.g. the Mayor vs. the K-12 educationalists). Different tools were needed to be written for heterogeneous
data sets for different viewers. This induces a ‘silo’ structure in the enterprise model and enabling software.

4. Changes to model by one group could not be easily incorporated by another model engineer or any board member. They weren’t documentable because there is no generic format of a change to any specific flavor of the STEM Model. Similar situation was faced when different audience wanted to relate their work together.

- **GreenCity Energy Management Simulation**

  This simulation has high decision and policy making ability, but subject to an assumption that the energy input modeled by the software is indeed realizable. In addition to this, none of the aftermath of drawing this energy input is being modeled in this software. An example of this is demonstrated at the end of this section, after we identify the problems with all the domain software introduced in this chapter.

- **HealthGame**

  The framework used for implementing HealthGame helps establish a sense-respond framework for specific scenarios, but does not allow any interaction between different scenarios. Consequently, maintaining traceability of decisions across scenarios is a challenge still unaddressed. This leads to limited learning and decision making ability, and limits the application of the sense-respond framework despite of its other advantages‡‡‡. Also, dynamically updating the goals and user status depending on either the decisions of another user, or other environmental factors that a user currently exists in, is not possible.

- **RealtyHub Live Search Application**


‡‡‡ The key advantage is enabling a sense-respond framework with high user outreach for participatory services. Other features are provided in [32].
Similar to earlier examples, this application behaves like a static data store if it only processes costs and taxes. Any user thinks of factors apart from monetary payments. Modeling these factors is essential owing to the advances in technology and interconnectivity. Users can exploit the web 2.0 concepts and social media, at the least, to obtain lot of information regarding anything tangible or intangible. Providing the right information to the user has high ability to affect business profit in the current market, particularly as many systems, like Facebook, Twitter, and RSS feeds as few examples, work to remotely provide latest information to the users.

However, identifying what information to provide and how to provide it in RealtyHub, is not trivial. Interfacing RealtyHub with “all” the software out there in the world is impossible, as is for any software.

The Lost Real-World Domain Inter-Connection:

This subsection demonstrates a scenario of interconnect in the real-world domains of the above software, which is impossible to model in their as-is or any enhanced but isolated implementations. For example, consider the following set of user decisions:

- User simulates an energy usage for Columbus, Ohio using the GreenCity software
- The power station that serves Columbus is in Chicago. The simulated energy input increases the load on a power station beyond its efficient operational capacity:

As a result, following would occur in real world:

- The supply-chain for that power station will be different
- The power station will start polluting because of crossing its efficient range of operation
The whole ecology around the power station will be affected, and hence the health of people in Chicago will be affected.

Less people will prefer living or buying properties in Chicago, people would be alerted of the fact that Chicago is now being reported as a polluted city – either through social media or by word of mouth. Conveying this information to the user would be a moral and legal responsibility of the government, if it has to operate the loaded power station and let it pollute more.

However, there is no way this aftermath can be modeled in any of the independent software. The domains need to be related to establish interrelation between their elements like objects, interactions, decisions/learning or knowledge. The example scenario presented in this section is used to validate the operation of the proposed framework.

Following section presents the characteristics that make the domain software presented in this chapter sufficient for validation purposes of the proposed framework.

3.6 Sufficiency as Validation Examples

We use the software applications described through sections 3.2 to 3.4 for validating the operation of the framework. In this section, we argue the sufficiency of these software applications as substantial example for the operational validation. As described in section 3.5, each of the software applications described in sections 3.2 to 3.4 have the following characteristics that make them a typical isolated domain software, and hence useful for validation of this research. Each of the domain software presented in 3.2 to 3.4:

- Works in a localized environment & data set, and represent user interactions in some constrained domain.
- Models a part of reality independently, has its own view of the real world
• Provides some learning abilities and enables a decision making ability to the user

• Has dynamics, if any, based on an object interaction model close to real world interactions

• Does not inherently support or exploit re-use, and any user decisions and derived knowledge is not sharable

Following chapter describes the framework proposed to enable interconnect mentioned in section 3.5. This interconnect is not just at the object or implementation level, but also within the individual user interactions (user decisions) and the derived knowledge.
4. The Cyberinfrastructure Framework: Mirror

This chapter starts with a conceptual description of the solution framework, and then delves into its architecture and design details. It then presents some of the validations for the framework design, as mentioned in Chapter 2.3. Other validations are presented in a later chapter when we introduce all the implementations and framework operation. The later sections in this chapter present an implementation of the core framework component, and provide a guideline for development and configuration of a Scape with the Core component.

4.1 Introduction to the Mirror Framework

We name the solution framework as “Mirror” owing to its goal of mirroring, via simulation, all the real-world interactions and knowledge for mining and decision making. The inspiration for this name is derived from [57]. This “mirrored world” ultimately represented by Mirror is co-developed by different users collaboratively. This section explains the framework concepts.

**Mirror Core**: Mirror Core is the software that stores the shared upper ontology for representing shared portion of the underlying reality, has knowledge of all the components working in the system, and can communicate with all these components. It also provides services to the Scapes to exploit and share knowledge across framework. Imagine this as the earth where developers build on the ‘basic things’ (analogy: land, water, air, minerals, etc) to develop various usable complex objects, processes, decisions, etc (context/domain analogy: industries, cities, organizations, processes, etc) to achieve some purpose. Mirror Core is the enabling component of the whole framework because it helps integrate the domains, knowledge and compose services using sharing and re-use. The Mirror objects modeled in Core have attributes that include spatial, visualization, role, services, time metrics, current state and history, etc. We talk more about mirror core in section 4.5.
**Scapes:** Scapes is software that enables object and interaction in a particular context or domain in real world. Scapes are conceptually defined as developer-defined added or refined functionalities onto the mirror core. Technically, Scapes are a particular configuration of: 1) roles, 2) entities like assets, humans, automated software and resources, 3) events, 4) requirements, and 5) interactions. E.g. Business management ‘Scape’ has finance management ‘subscape’ and has many ‘users’ playing different ‘roles’ for a common purpose of managing business. Scapes cannot arbitrarily manipulate the mirror assets they use. To configure a Scape in the Mirror, we follow a set of prescribed guidelines to define the Scape objects and dynamics. For example, to win scavenger hunt, user must find X clues at {A1-M1} places and report them to the jury Y before time T on day D. This Scape will involve assets and clues at different locations, roles, users, interactions, events, etc. Naturally, Scapes can have sub-Scapes. Scapes are classified into two major categories:

- **Expansion Scape**

  Expansion Scapes do not necessarily have a context of operation. These can be just a set of services over some data that the user can use regardless of any operational environment, or that the usage of those services is not subject to a fixed set of law or rules enforced by the local context. Example of such type of Scapes could be a data warehouse extension to the Mirror Core that provides different views of the data to the users. E.g. ODJFS Scape implemented on the Core.

- **Simulation Scape**

  Simulation Scapes, on the other hand, do have a context or environment of operation governed by some laws as enforced by the Mirror core as well as the local Scape intelligence. Such Scapes usually tend to be more interactive with the users, by providing a simulation type functionality that simulates some subset of the real world interactions of the users. Example of such Scapes could be typical gaming applications where users play the game in an environment subject to some rules within the game. E.g. HealthGame implementation on the Core, World of Warcraft, Age of Empires, CityOne by IBM, University Scavenger Hunt, etc.

Figure 2 shows a snapshot of how the Core and Scapes work together.
The Scapes share their operational data (user decisions, interactions) and derived knowledge with the Mirror Core. The Core transforms this data into its own vocabulary using the mediation services over the shared upper ontology. Once the data is transformed into the terms of shared upper ontology, the Core can apply all the rules modeled by the Core and individual Scapes to make explicit the implications of interactions between Scapes. The implications and knowledge this developed by Core is made available to the Scapes; but it the Scape developer's discretion to utilize it in the Scape dynamics or not.

The following sections detail this operation of the framework. Section 4.2 starts this with introducing the design and architectural considerations for the framework. Section 4.3 describes the ontology specifications used in Mirror. Section 4.4 presents the Mirror vocabulary, i.e. the shared upper dynamics ontology, and section 4.5 describes the reference
architecture of the framework that is used to implement the Mirror Core and Scapes. Section 4.6 presents the process of developing Scapes with the framework.

4.2 Architecture Approach

This section introduces the architectural considerations and decisions for the framework.

Model Driven Architecture

Models are used to provide abstractions of any physical system in order to allow engineers to reason about the system by ignoring impertinent details and focusing on pertinent ones. All forms of engineering rely on models to understand complex, real-world systems. As OMG§§§§ defines, “MDA is a way to organize and manage enterprise architectures supported by automated tools and services for both defining the models and facilitating transformations between different model types.” Based of OMG’s standards, the MDA separates business and application logic from the underlying platform technology. Platform independent models of an application or an integrated system’s business functionality and behavior, built using OMG standard specifications, can be implemented on any specific platform. These platform independent models document the business functionality and behavior of an application separate from the technology specific implementation details. This insulation helps achieve following major goals through architectural separation of concerns:

- Portability
- Inter-operability
- Re-usability

More documentation and information about MDA and its specifications can be found on the OMG site and the MDA guide. *****


**** : OMG MDA website –[ http://www.omg.org/mda/ ]
The concern we address using MDA approach is an optimal separation of domain and system understanding, functioning and representations, as explained in the next sections.

**Ontology: Knowledge Expression & Integration of Domain Understanding**

A common, machine-understandable and standard representation of the critical system objects (static and dynamic) poses as an utmost priority in order to bring together different stakeholders onto the same goal and evaluate the processes from their own viewpoints. RDF Schema based on the RDF specifications provided a way to describe an object in relation to other objects. Clearly, just representing the knowledge objects did not serve the purpose towards discerning information from the physical data. W3C realized the need to represent higher level associations between objects and their properties, and set a working group (called Web-Ontology Working Group) in 2001 to create a more expressive ontology language. The first version of this language, called as OWL, was released as formal W3C recommendation in 2004; and one of its major enhancements called as OWL2 became a formal W3C recommendation in 2009. Both OWL and OWL2 are based on RDF specifications for their formal representations on any computer. We use this (OWL2, but hereafter referred to as OWL) concept of ‘ontology’ specification in Information Science to create a shared understanding of the domain objects and functionalities at different levels [14, 15].

Though OWL specification is a way to express the semantics of the domain, but it is not essentially a language and it does not have a full expression capability bound by predictability of inference. On the other hand, Object Oriented Programming is a language but is not semantically as rich and expressive as OWL. Naturally, there has to be a merger between these two representations and interpretations. Striking the right balance in describing and realizing with these two technologies and using the right transformations from OWL to OO [1, 2, and 8] will be achieved by interfacing with external tools. We use OWL/RDF to

††††† W3C: World Wide Web Consortium (http://www.w3.org/)

‡‡‡‡‡ owing to their inter-compatibility, and the feature usage that dictates what flavor is being used. More information on RDF, RDF Schema, OWL and OWL2 can be found on the W3C website
describe as much static information as possible, while also allowing dynamics to be represented at the discretion of developer.

Ontology Types & Examples:

Domain Ontology: Domain ontology models a specific domain or part of the world. It represents meanings of the terms as they are to be interpreted in a particular context. E.g. modeling of “bat” as an object used to hit a pitched ball in the Baseball context, but as a flying nocturnal living thing in the living beings’ context.

Upper Ontology: An upper ontology models objects that are common across domain ontology. E.g. Person is a Living Thing, all Living Thing are Things, etc.

There are many standard upper ontologies defined that describe very generic natural language terms that occur across the world, e.g. Cyc or a taxonomy like IPSV [34]. There are also ontologies that are hybrid [12], which go few levels up the abstraction from typical domain ontology. Few examples of this are the Ecosystem Ontology, Bio-medical Ontology and ResearchCyc [13].

Our framework uses ontologies for defining the domain objects [28], mapping the domain objects to the system (framework) objects [2, 8, 1], and defining a hybrid ontology that serves as a localized, evolving “upper ontology” to relate the domain applications [11] with each other and to a common goal execution, or enhancement of the framework itself (the innovation loop). This component will help the integration [5, 18] of heterogeneous elements in the framework implementation.

Rules & Inference

The rules mainly refer to business rules that are typically a part of any system use case. These may be directly related to a set of objects’ behavior or any event in the system operation or any generic law that is to be established unconditionally. Such rules can be modeled using many designs and applied for inference, or evolution [31, 9]; however, the current
framework prototype limits the types of rules allowed. This is discussed further in section 4.4. The rules in the system should also be designed to be in a context of the agent rather than individual agents in the Agent Modeling terminology, as advocated in [17]. Analogous to this in our framework is the segregation of rules that apply to the underlying commonality, from those that are domain specific.

Configuration management, Content Management

The framework creates and maintains many work-products that are maintained using typical set of software configuration management operations. These work-products include various ontology definitions, references, object definitions, domain services definitions and service registry. An important feature of any cyberinfrastructure is to maintain traceability and ownership information about as much work products as possible.

Product Line Architecture & Software Evolution

Product line is a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs and that are developed from a common set of core assets in a prescribed way. Product line architecture refers to a description of structural properties for building a group of related systems (that is, a product line), typically the components and their interrelationships. The inherent guidelines about the use of components must capture the mean for handling required variability among the systems (sometimes called reference architecture). We use this concept for designing the mirror core and Scapes with the core.

Meta-data and Meta-Programming

The use of ‘meta’ refers to abstraction. Abstracting data can be of great help for tagging, semantic analysis and mediation purposes. Two features of Meta-programming are exploited here. First is to expose the internals of the run-time engine to the programming code through application programming interfaces (APIs); second through dynamic execution of
strings that contain programming commands. We use these abilities to mediate the data and represent process commands in the framework.

**Policy Languages**

Policy languages are a great tool to represent the system wide governance, access and other security policies, as well as for representation and mediation of simple data across the system. Policy languages play a significant role in enabling inter-operability. We use policy languages to specify service definitions and usage (web services), security policies, possible open licensing and extensible markup language files.

**Designing (for) a Cyberinfrastructure**

As introduced in chapter 1, the scope of software this framework enables is of the scale of typical cyberinfrastructures. Consequently, the framework had to be designed keeping in mind the environment and issues that a typical cyberinfrastructure is subject to. This environment and the set of issues are studied using the references mentioned in section 1.3 along with some real world examples of cyberinfrastructures. This study and the lessons learnt are described in detail in Appendix C in order to avoid digressing from the current discussion. This study motivates the crucial design considerations of extensibility, flexibility, interoperability and evolution for the framework implementation. How these considerations are being addressed in the framework is explained in section 4.4 and 4.5.
4.3 Mirror Ontology Specifications & Mappings

Mirror/Scape-Upper Ontology for the Scape Domains

The Mirror Upper Ontology is goal driven (goal in user’s perspective) schema showing generic user goals in real world and how those are mapped to the Scape implementations in the system. At its instance levels are the Scape names that may or may not be currently implemented, and up the hierarchy, multiple Scape implementations could be satisfying one or more goal and can be thus linked to one or more higher level goal in the Mirror Upper Ontology. A simplified subset of this Ontology is depicted below for illustrative purpose:

All ontology specifications and mappings in the framework are shared throughout the framework.
It can be seen as a classified set of contexts in the working world, in which different set of interactions take place amongst different agents playing different roles, together towards a goal in that context. This ontology helps users develop and discover possible Scapes and services using a fixed classification and organizational scheme, by representing the currently known world as it is mirrored. Such ontology, as mentioned earlier, is derived from the generic upper level ontologies like Cyc, SWEET, etc.

**Shared Upper Dynamics Ontology for the Scape/Core**

This ontology for Core/Scape describes all the objects, their properties and the dynamics. This establishes an understanding of what object data does the Core/Scape own and work on in the system. The objects are related to each other using a set of defined relations. Instance data of such objects can also be included in this ontology to describe some dynamics. For example, a Scape might have a <person> as an object with properties <name>, <address>, <social security>, etc. and relates to <father> by relation <childOf>.
This ontology is typically derived from object interaction diagrams and domain models for the Scape, while it is designed separately for the Mirror Core as described in the section 4.3.

**Domain Mapping Ontology**

This is a schema that describes the classification and inter-relation of Scape objects with relevant other objects in the Mirror Core. Each Scape has zero or more DM ontology files explaining relation of that Scape objects with other Scapes, and/or Mirror’s objects. All such files for the respective Scape are stored in a location that is centrally accessible to Mirror system to enable services on the ontology files.

Figure 5 shows a sample mapping of the domain ontology in GeoGame [33], a generic Scavenger Hunt scenario, and Healthy Living scenario [32] with the Mirror Shared Upper Ontology for Dynamics. The left-most column in the figure represents objects in Mirror Shared Upper Dynamics Ontology, while the rest of the columns represent the corresponding domain objects mapped.
The domain-mapping ontology, developed by domain experts and Scape developers together, specifies the context of operation in higher detail than the Mirror Upper Ontology for Domains, and is the interfacing between the Scape’s modeled domain and the mirror implementation elements. As an example, in a medical domain Scape, say Health Care Record Management, “patient”, “Record Documents”, “Hospitals” (physical entity), “Department” (physical entity, if present), etc map to Assets, “Nurse”, “Doctor” map to Roles, “Add New Record”, “Delete Record” etc are Interactions through the Roles that enable some operations in the Scape, and so on. The domain-mapping ontology also maps the possible conceptual sub-grouping in the domain, e.g. a set of Assets, Roles and Interactions form a “Department” in a “Hospital”.

Figure 5: Examples of Domain Mapping for Scape Objects

****** Mapping is with the Shared Upper Dynamics Ontology in Mirror Core
Shared Service Ontology

Service ontology relates the services of a Core/Scape with each other, or as using some other services in the Mirror, by using a set of predefined relations in Mirror. These include, for example, binary relations like “GISEnables”, “ImportsService”, “Invokes”, or unary tags like “SearchServices”, “Analytics”, etc. This ontology helps establish relations between two services within or across Scapes, and also tag services based on their functionality. This component will help describe behavior of the Scape depending on the richness of the pre-defined relations and tags in the Mirror, and enables service discovery, composition and re-use in the Mirror framework. Service Ontology can be derived from use cases, design and the services identified for a Scape.

Scape API

Scape API is a configuration file created with a custom pre-defined syntactical structure, as provided in section 4.6, to describe the Scape in a programatically declarative fashion. It collectively describes the Scape objects and behavior; it includes all the above ontologies and can include any extraneous descriptions of the Scape at the discretion of the Scape developer. Using this description file, it is possible to aid (automate) the development of a new Scape, and ideally achieve a degree of simulation of the run-time behavior of the Scape.

The Scape API configuration and ontology files for all the Scapes in the system are stored or linked in a location accessible to all components in Mirror. The Mirror Core enables Search and Import/Export services on these specifications to dynamically and declaratively create new Scape configurations. As with ontologies, the Scape API file has to be specified for the Mirror Core as well, along with the other Scapes.

Utilization of this Scape declarative specification files is done in two ways:

• To help declarative Scape development

Using frameworks like ActiveRDF, the RDF descriptions are transformed into Object Oriented code in different languages like Java, Python, etc using Factory pattern, thereby providing some CASE type ability. Note, however, that the current
prototype requires manual intervention owing to incomplete syntactical grammar specification and custom parser and interpreters for the specification files. Interfacing with existing tools can be done once the grammar is frozen. The algorithmic process that the tool would employ to process a Scape API configuration will look like:

- Load Scape API file into tool
- Preprocessing all the directives (e.g. @, ##, other markup entries)
- Parse, tokenize and synthesize the file elements using a compiler
- Import Processes – uses Mirror’s import object schema and import object data services, creates the derived object schema(s) for the local Scape.
- Rules –
  Jess rules -> consolidate (from all sources) Jess rules into a file
  Owl rules -> consolidate (from all sources) OWL rules into a file
  ETP rules -> generate sample if-else like structure to expand, based on the instances used to defined these rules in the ontologies. This will also help maintain completeness of the model of the Scape system, and consistency of its implementation with the declarative definitions in the Scape API.

This process, however, is not automated in the current prototype implementation of the Mirror, but is described and presented in Chapter 8 as it forms an interesting direction for furthering the framework in immediate future.
By an analyst/programmer/Scape developer by locating different services, Scapes and objects in the current Mirror system – to enhance it by building something comprehensive.

When a Scape is being developed, it can refer to the existing objects and services in the mirror system through other Scape’s API description files. Since the Scape API describes the Scape, and the Domain Mapping Ontology describes the mapping between Scape and mirror objects in the system, we can run a typical rule engine (e.g. Top Engine) or just an OWL DL reasoner to infer and update the upper ontology which expresses how Scapes are related to each other and the Mirror-Upper ontology. On an updated Upper and Domain-Mapping ontology, the Scape developer would use search and query services through the Mirror system.

In the next section, we describe the Mirror Shared Upper Dynamics Ontology which is the key component that enables the framework operation, through the Mirror Core.

### 4.4 Shared Upper Ontology for Scape/Core Dynamics

The shared upper dynamics ontology serves as a way for the Mirror framework to mediate any representational or semantic differences across multiple Scapes, tools and data sources. Inferring of inter-domain knowledge and implications is done using this shared upper ontology. Following are the elements of the shared upper ontology in the Mirror framework:

**Roles:** Capture the ‘who’ and is the description of a specific type of user. Users can be categorized into different roles based on their responsibilities, history, or authorization level. Roles can even be made of other roles. Farmers, market, legislature can be seen as examples of roles. In the real world or during a game simulation, an entity or player plays different roles. Interactions may often require multiple entities in specific roles. In general, if the entity/user is assigned to a role, and the user is authorized, the user can participate in an interaction.

**Assets:** This concept captures, in the static view, the 'what' is used or affected due to an interaction. Assets can be grouped into different groups, and can have a many-many
relationship between assets and asset groups. An asset can ‘contain’ other assets with the following relationships:

- Aggregation (e.g. a set of marbles)
- Composition (e.g. a car is composed of engine, wheels, controls)
- Containment (e.g. a room contains a chair, a jar contains water)

Assets in Mirror framework can be buildings, land, cattle, cars, buses, stadiums, pens, etc – anything that can be seen as tangible objects in real world. Assets are modeled to include real-world attributes like GIS coordinates, owner, status, is being used or not, orientation, motion, etc. As we plan to generalize and move from just supporting gaming interactions in the prototype to include real-world execution scenarios in the final framework, the schematic definitions of the assets will evolve to incorporate more attributes than what are currently modeled.

By default, the Mirror assets are shared amongst all users, Scapes and events. However, a view of real world can be captured and locally modified within a Scape, but it is important to note that all the assets are kept consistent with the mirror core, and not a particular Scape. This dictates if a Scape has published-subscribe or publish-publish relation with the Core. E.g. a Scape copies GIS data of assets in 8th Block between High & Neil Ave and represents it in graphical 3D form. It allows Civil engineer users (or users with certain role in that Scape) to restructure the assets, introduce new assets in a way to minimize travel/congestion in that block. If the changes are approved by some approval authority in that Scape, those are actually proposed and the (mirror) could be changed to accommodate those. Other Scapes will also use the new block structure when our Scape publishes its changes to the mirror – a classic example of aiding decision making.

**Agents:** The idea of ‘agent’ is in the runtime perception, and serves the same taste as in typical Agent Based Modeling. When assigned to a role to perform an interaction, the active entities are called agents for that interaction. Agents are run-time entities and are modeled in Mirror through their components viz. Interaction, Roles and Assets or Users playing those Roles.
**Events:** Capture the ‘why’ and ‘when’ aspects. An event is any 'happening' in the world known to Mirror, which can cause an asset to change its state through associated Interactions. Events can be random as in a 'fate event' or 'scheduled' (e.g. every five hours). In a simulation, the events can be contracted or expanded to different time scales (a second in the game can constitute a day in the real-world. Note that events here do not refer to all the lower level system operations, but a user “clicking” a request that is a purchase request is modeled as event. This is because actions in the Scape that lead to some goal modeled within the Scape are crucial to understand the Scape operation and user behavior. Hence, the operational events that directly relate to the software use case of the Scapes are modeled as events in Mirror. Some examples of modeled events are earthquakes, floods, accidents, etc. These are random events that are not in anybody’s control though they have some consequences that affect the mirror world. Scheduled or Non-random events include user defined events such as hosting a concert at a stadium, or holding a racing competition, etc. These events also have some consequences that in turn might cause some other events. E.g. Concert might cause traffic jams/accidents, which will need re-routing of normal traffic routes. An event is sometimes also referred to as a request to be served.

**Interaction:** Captures further the ‘what’, ‘when’, 'where', and ‘how’ the event changes the entities. An interaction is initiated by a request (event) from an agent. The interaction processes are applied after this initiation only if certain requirements are met, as we see in the Rules section. An interaction is executed or implemented by agents that apply processes that change the state and attributes of other entities. Note that we use the word 'Interaction' to also mean any in-the-small collaboration among agents (e.g. a customer, a provider, an owner) that cause a state change in an entity (e.g. asset transfers, purchases and sharing). An agent can be a user, asset, interaction or an event.

The following diagram shows example relationships between the conceptual elements discussed above. These concepts and relationships form the base of shared upper ontology for the Mirror Core. The complete shared upper ontology is provided in Appendix B.
Rules and Rule Execution: The consequences of an event are modeled using event rules and executed using a rule inference engine. The event rules are conceptually represented in the form of triplet \(<event><requirement><interaction>\). An interaction can itself be in two parts - execution processes and delivery processes. That is we really have \(<event><requirement><execution><delivery>\) as an \(<interaction>\), derived using concepts from [30]. These parts can be executed in different systems.

\(<Event><Requirement><Interaction>\) triplets are the rules that are executed dynamically. For example: if a storm fate 'event' occurs and the 'requirement' of a probability level holds, then the damage 'interaction' rules are applied to protect certain roles and assets. Another example is if a player UI 'submit event' occurs and the 'requirement' that the player is authorized is met, then the player can perform certain 'interactions' like sell an asset and rules update the assets

†††††† The complete shared ontology is provided in Appendix B. The database schema for Core is derived from it. The completeness of this ontology as a “shared” ontology in our context is discussed in section 4.4.
**Event Triggered Procedures (ETP):**

Rules in any Scape can be of two types:

- **Access Control rules:**
  
  E.g. role based access control. The roles need to be modeled in OWL, to be able to:
  
  - establish relationships between roles and other objects in the Scape
  
  - relate to the upper ontology, which is in OWL, easily

- **Event-Triggered Procedures (ETP):**
  
  Event-Triggered Procedures are the rules that are preceded by some pre-condition in their execution, and their execution depends explicitly or implicitly on the precondition and the trigger source, either external or internal to the Scape. These rules are found in code in model classes usually in if-else or similarly structured control blocks, or crons or daemon processes running in background.

Both these type of rules can be represented using the event-condition-action triplet.

E.g.:

- **Access Control Rule in OWL:** If a user1 is OSU Faculty, if user2 is OSU student, and if user1 advises user2 then assert “user2 is in user1’s workgroup”

  In the event condition action triplet form, this can be represented as:

  Event: Execute the OWL reasoner with the OWL file and rules
  Condition: user1 = OSU Faculty, user2 = OSU Student, user2.advisor = user1
  Action: Add user2 to user1.workgroup, and therefore it will have access to whatever data and tools the workgroup is assigned access to.

- **ETP type rule:**
  If (SensedInteraction)
{ if (type = “GIS location Update”) { suggestNearestCoffeeShops(); } }

Or another example could be a simple cron:

```bash
## Run on every Sunday at 5pm
00 17 * * 0
If ($Environment.user == administrator)
{ /home/admin/bin/AnalyzeReports() | mail -s “Reports” admin@domain.domain }
```

In the event-condition-action form, for example, the first rule can be written as:

**Event:** Interaction is sensed

**Condition:** interaction type is GIS location update through GIS chip on a device

**Action:** Execute a method that suggests coffee shops around the new GIS location

For second rule:

**Event:** Today is Sunday and time is 5pm

**Condition:** If admin is logged on in current session

**Action:** Analyze the reports and mail him/her the copy of its results.

**Scape Rules:** We envision multiple interrelated real-world collaborations i.e. Scapes interrelations, within the Mirror CI, using Scape rules. All objects are either local to a Scape or global (i.e. shared) among all Scapes and thus a part of the Core Mirror as shown in Figure 8. The local Scape objects can be views of the global ones or unique to the Scape. Local Scape-specific views of asset configurations can be temporarily inconsistent until published to the Mirror - the primary record. The rules are represented using a markup language for rules directly relating to OWL defined objects, and using specific destination technologies like JESS (Java Expert System Shell) for types of rules that involve external actions like communications, or need to represent instance specific conditions like numeric inequalities, and so on. The combined application of OWL and other rules like JESS, or SWRL is a topic of research in itself. The key tradeoffs, however, are expressivity against decidability. The interfacing between JESS and OWL is deferred to future work; however, OWL is used extensively to express the objects and dynamics within the Scapes and the Core.

**Affordances:** Different assets may have different affordances that can be explored collaboratively in different Scapes and this is permitted by the local Scape views. The Mirror
enforces consistency between local and global objects through publish-and-subscribe rules. In addition there are certain constraints local to each object identified above, and certain ontologies and domain rule sets that are used in the application of specific Mirror rules. Finally, there are external sources of data that are federated and third-party functionality (e.g. Google earth) that are also used. The Mirror scope rules allows different discussions in different Scapes based on different views of underlying assets and shared resources in the physical world. For example, a school bus asset can play a role in a 'Student Transportation' interaction under normal circumstances but can also play an 'Emergency Vehicle' role during a hurricane. Roles help achieve different outcomes by exploring affordances of an asset in different interactions and Scapes.

4.4.1 Design Validation: Completeness of Shared Upper Ontology

The Shared Upper Domain Ontology is dynamically extensible in the system without any repercussions. This is because it does not directly relate to any implementation constructs in the framework, but provides an understanding of how the domain goals relate to each other at a high level.

The completeness of Shared Upper Dynamics Ontology has two aspects to it:

i. Representing the real-world objects and their attributes

ii. Representing the dynamics of the real-world interactions

This ontology, as implemented in the scope of this thesis, is certainly not “exhaustively” complete. As mentioned earlier, we use OWL to express the objects and interactions. So, we are limited by the descriptive logic that OWL possesses. OWL, adopted as a standard by W3C is very expressive as it serves whole of the current semantic web. However, OWL is not a programming language, so it does not have the ability to “process commands” dynamically. An example where this is needed is an interaction that has after effects like inter-system communication, or a need for built-in functions to do some repetitive processing. All such functionality is currently embedded in the code surrounding the
ontology. Using a rule engine with OWL expression and inference ability will greatly help increase expressivity of the system, as well as neatly structure the code for better maintenance. Another example where rule engine helps is in representation of numeric inequalities instead of conceptual entities. However, the interfacing and incorporation of rule engine ability with OWL is a subject of different research. This is because the two have to be carefully balanced to strike a trade-off between expressivity and decidability. Some attempts at striking such trade-offs are explained in [60] and [61].

So, as the existing OWL reasoners model more and more SWRL (Semantic Web Rule Language) capabilities, the expression and inference ability of the Shared Upper Dynamics Ontology will be improved. Pellet is an example of such reasoner that is incorporating SWRL features. However, the Shared Upper Dynamics Ontology, as it is in the scope of this thesis, is designed to allow most real-world interactions, objects and scenarios to be expressed. To argue its “sufficiency” at this time, which would lead to “completeness” during evolution of the framework, we demonstrate that many real-world interactions, games, and simulations can be expressed using this very ontology. These interactions include transactions, communications, administration and governance, collaborations for competition, and university or school interactions. Following is an example of how the objects and interactions in Second Life game can be expressed using the Mirror Shared Upper Dynamics Ontology.

<table>
<thead>
<tr>
<th>SECOND LIFE</th>
<th>MIRROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar appearance</td>
<td>asset</td>
</tr>
<tr>
<td>Asset</td>
<td>asset</td>
</tr>
<tr>
<td>Shape, texture</td>
<td>assetAttributes, assetState</td>
</tr>
<tr>
<td>Group</td>
<td>group</td>
</tr>
<tr>
<td>Friend</td>
<td>userRelation</td>
</tr>
<tr>
<td>Agents</td>
<td>users</td>
</tr>
<tr>
<td>Permissions</td>
<td>permissions, role</td>
</tr>
<tr>
<td>Objects</td>
<td>assets</td>
</tr>
<tr>
<td>Images</td>
<td>asset</td>
</tr>
<tr>
<td>Motion</td>
<td>presence</td>
</tr>
<tr>
<td>Orientation</td>
<td>presence</td>
</tr>
<tr>
<td>Events</td>
<td>events</td>
</tr>
<tr>
<td>Sounds</td>
<td>assets</td>
</tr>
<tr>
<td>Transactions</td>
<td>events, interactions</td>
</tr>
<tr>
<td>Chat</td>
<td>message</td>
</tr>
</tbody>
</table>

Figure 7: Domain Mapping for Second Life on Mirror
The interactions in Second Life are expressed in the same way as shown in Figure 6. Similarly, the objects and interactions in GeoGame are expressed as shown in the following figure 8. The Entity is mapped to Asset, while other mappings are obvious through naming of objects in the figures 8 and 9.

![Diagram of Domain Mapping for GeoGame on Mirror](image)

**Figure 8: Domain Mapping for GeoGame on Mirror**

The object and interactions for HealthGame are represented in Mirror as shown in figure 9. The Interaction in HealthGame is mapped to Interactions in Mirror, while the Agent is mapped to User in Mirror.

**** Figure borrowed from [33].
Few other scenarios that we modeled using the Mirror Shared Upper Dynamics ontology are scavenger hunt scenario, and student interactions within a university.

We conclude that this expressive power of the Shared Upper Dynamics Ontology is sufficient to demonstrate various domain interconnects for the validation of prototype framework in this thesis, because it was able to –

i. Represent the distinct, specialized domain objects
ii. Allowed its mapping to the generic “upper level” objects stored in Mirror
iii. Represent the dynamics in the disparate domains of above example games and scenarios.

Figure borrowed from [32]
Moreover, this ontology can be extended to include more expressive power and utilize more OWL features as this framework evolves. The flexibility of the framework, as discussed in following section, allows us to keep the framework components updated with the changes in its ecology.

4.5 Framework Reference Architecture and Design

This section introduces the architecture used to develop the framework in order to leverage the shared ontology with the domain objects and interactions occurring in different Scapes. It starts with a high level approach, and then introduces the software architecture used to implement the framework, along with its software qualities and trade-offs hit during the design.

The framework reference architecture is shown in figure 10. At its bottom-most level in the figure are databases with persisted entities (objects, interactions, decisions and knowledge, ontology components, etc). Mirror Core provides services over these entities to utilize them; these services are introduced in section 4.6. The operational data of the Scapes is transformed into Mirror Core vocabulary using the Shared Upper Dynamics Ontology, the Domain ontology, Domain Mapping ontology and the mediation services. The Scape relates its objects like events, interactions, roles, etc to the Core’s objects, and describes any relationship between the interactions occurring in the Scape and the interactions modeled in the Core. The mediation services help transform actual data available for these conceptual entities, and also maps data that is not directly related to the Core through domain mapping ontology.

Details about use cases, implementation, Database Schema can be found in Appendix A

All the modeling for the framework was done using UML notation
The Mirror Core runs its services to apply all the specified rules over the collected data to update the shared ontology specifications with instances from the data shared from Scapes, and to infer new knowledge from that data. This inferred knowledge is stored as axioms and can be relayed back to the Scapes using the mediation services, if the Scape wishes to receive it. Currently, the Scapes have to manually “pull” the inferred axioms back to update their dynamics; this process is not automated.
The architecting process and decisions were based on a value-based approach as in [29], exploring different patterns and components/technologies to be used for the framework. The design is currently foreseen to follow the typical road-map for SOA Evolution [35]. Along with the approaches described in section 4.2, following architectural patterns are incorporated in the current architecture:

i. Service Oriented Approach

The services in mirror are currently implemented using the web or remote services framework (either SOAP or AMF) and uses HTTP and XML for communication, UDDI and WSDL for service discovery and specification, in addition to the framework ontologies. Usage of REST web services is something of interest since it will help the framework scale better and optimize the performance by replacing costly redundant web service calls by simple data transfers across web services. But the choice of SOAP over other technologies like JMS was because SOAP allows more flexibility [58].

ii. Model-View-Controller (M-V-C)

The Core adheres to the M-V-C architectural pattern, and uses different decoupled technologies at each tier. This allows adaptation to new requirements and modularization, and thereby helps achieve flexibility and reusability. The Controller and View are implemented using the well-known Adobe Flex framework, and the Model operates in J2EE (Java) environment. The enabling technologies currently used in each tier are as shown below.

‡‡‡‡‡‡‡

The Mirror core is designed [3] using the J2EE technologies, M-V-C architecture in conjunction with OWL/RDF technologies to serve the non-functional requirements of flexibility, extensibility, interoperability and scalability. The framework is based on an EA - ACE idea [30] and supports evolution through the concept of ACE fractal patterns.

Below is a description of the technologies used and their purpose in the framework.

**Java EE (3.4):** For writing all the core code, using eclipse (Ganymede) IDE for writing Java as well as Flex code. This works as the Model in the M-V-C for the Core.

**Adobe Flex 3:** The frontend development component for Rich Internet Applications (RIA) like the UI of the Core.

**BlazeDS:** For making Java methods remotely accessible over the HTTP port. Most of the services are called using BlazeDS since it provided faster response than the standard SOAP/XML based web services. So we write java classes in our backend and remote them through BlazeDS so that the Flex frontend can call/execute these methods.
**JBoss:** as the Application Server to forward requests over HTTP to appropriate components

**JPA:** as the Persistence Layer to enable easier data manipulation and persistence (bundled with JBoss)

**PureMVC:** A MVC framework for Flex applications to make the client code clear/structured and tiered.

**SVN:** Repository for collaborative development and version control

**MySQL:** Relational database to store all the data.

The block arrows represent the interfacing with rest of the non-model Core components, while the red ovals represent components/services currently not implemented. A list of implemented and not implemented services is mentioned in section 4.6.
The following subsections discuss the qualities of this design enabled by this architecture and different design patterns exploited during the framework implementation.

### 4.5.1 Key Design Considerations & Solutions

“A design pattern is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.”

Design patterns speed up the process of development, testing and also help to avoid common problems that have already been addressed through some template solution. It also helps improve the code readability, despite of incorporating indirections for flexibility.

We used the following design patterns while implementing the solution:

- **Creational**
  - Builder pattern: to separate construction of object from its representation, to help re-use same construction mechanism for multiple representations. This is used in creating Scape objects with different properties.
  - Lazy initialization: to delay construction of any objects in memory till the time they absolutely need to be constructed. This introduces a delay when the object is called for dynamically, but, on the other side, helps improve the startup time and keeps more memory free for computation purposes.

- **Structural**:
  - Adapters: Adapters help two objects work together that otherwise would not because of incompatible interfacing mechanisms. This increases

---

Borrowed from Wikipedia for introducing the concept of Design Patterns
interoperability with the framework. Mediation services use this to enable interoperability.

b. Bridge Pattern: to decouple an abstraction (specification) from its class implementation, allowing easy adaptation, and flexibility. We use this for the Shared Upper Dynamics Ontology objects.

c. Façade: provides a single point of contact to a set of interfaces in the system. We use this concept in the front end for the Core, with Flex framework, to make the framework easy to use.

- Behavioral
  a. Chain of Responsibility: to incorporate maximum de-coupling to allow flexibility, extensibility, and ease of maintenance.
  b. Command: to support enhanced operations by encapsulating request as an object. We use this for command processing invocations between the components in Core.
  c. Mediator: to define an object that abstracts the interaction of other objects. We use this to model the shared ontology as well as enable interoperability between disparate objects in the Mirror framework.

Section 4.5.2 summarizes this into the qualities that the framework inherits by implementing these approaches and design patterns.

4.5.2 Design Validation: Qualities of Framework Design

Summarizing from the earlier subsections in this chapter, the proven solutions provided by the architectural and design patterns help the framework achieve:

- Flexibility
• Extensibility, and

• Interoperability

• Security features currently implemented by the framework include production level firewalls on Linux, as well as security access policies deployed on technological component level. For example, the Flex front end allows remote requests only from known pre-configured domains in the Core.

4.6 Implemented Reference Architecture: Mirror Core

4.6.1 Ontology Specifications in the Core

• Building the Shared Upper Dynamics Ontology for Core

We extract the objects and their relations from the object interaction model for the Mirror Core, as shown in the figure below. The full ontology is provided in Appendix B.

![Diagram of Shared Upper Dynamics Ontology](image)

Figure 13: Building Mirror Shared Upper Dynamics Ontology
The boxes in the picture are objects and the connectors are the relations between the objects in the Core. The full model includes details to these objects and is expressed in a way that is intuitively transferrable to a database schema. It is derived from many case studies including Serious gaming software like GeoGame, HealthGame, Scavenger Hunt scenarios, Social Networking and collaboration software. The model isn’t exhaustively complete but suffices for the initial framework implementation since it encompasses the interactions taking place in many software and scenarios, as mentioned in section 4.4.1. Nonetheless, extensibility was a key feature kept in mind while designing the core. So any component of the Core can be changed or extended thanks to its modularized design. All this makes the current interaction model sufficient to evaluate the framework.

Following is the upper dynamics ontology derived from this interaction model.

```xml
<?xml version="1.0"?>
<!DOCTYPE owl2xml:Ontology [ 
<!ENTITY owl "http://www.w3.org/2002/07/owl#" > 
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" > 
<!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" > 
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" > 
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" > 
<!ENTITY MirrorObjects "http://mirror.cse.ohio-state.edu/2010/11/MirrorObjects#" > ]>

<owl2xml:Ontology xmlns="http://mirror.cse.ohio-state.edu/2010/11/MirrorObjects#" 
xmlns:base="http://www.w3.org/2006/12/owl2-xml#" 
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" 
xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#" 
xmlns:MirrorObjects="http://mirror.cse.ohio-state.edu/2010/11/MirrorObjects#" 
xmlns:owl="http://www.w3.org/2002/07/owl#" 
xmlns:xsd="http://www.w3.org/2001/XMLSchema#" 
xml:base="http://www.w3.org/2006/12/owl2-xml#" 
xml:stylesheet href="http://mirror.cse.ohio-state.edu/2010/11/MirrorObjects#" 
xml:stylesheet type="text/xsl" href="http://mirror.cse.ohio-state.edu/2010/11/MirrorObjects#" />
```
<owl2xml:Annotation owl2xml:annotationURI="&owl;versionInfo">
  <owl2xml:Constant>1</owl2xml:Constant>
</owl2xml:Annotation>

<owl2xml:SubClassOf>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;city"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;presence"/>
</owl2xml:SubClassOf>

<owl2xml:Declaration>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;event"/>
</owl2xml:Declaration>

<owl2xml:Declaration>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;group"/>
</owl2xml:Declaration>

<owl2xml:SubClassOf>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;hydroelectricPS"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;powerStation"/>
</owl2xml:SubClassOf>

<owl2xml:Declaration>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;interaction"/>
</owl2xml:Declaration>

<owl2xml:Declaration>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;interactionType"/>
</owl2xml:Declaration>

<owl2xml:SubClassOf>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;landArea"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:SubClassOf>

<owl2xml:EquivalentClasses>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;loadedPS"/>
  <owl2xml:DataMinCardinality owl2xml:cardinality="1">
    <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasProductionAmps"/>
  </owl2xml:DataMinCardinality>
</owl2xml:EquivalentClasses>
<owl2xml:Datatype owl2xml:URI="&xsd;float"/>
</owl2xml:DataMinCardinality>
</owl2xml:EquivalentClasses>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&MirrorObjects;mirrorUser"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects:user"/>
</owl2xml:SubClassOf>
<owl2xml:EquivalentClasses>
<owl2xml:Class owl2xml:URI="&MirrorObjects;pollutedCity"/>
<owl2xml:ObjectSomeValuesFrom>
<owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;hasPowerStation"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects;loadedPS"/>
</owl2xml:ObjectSomeValuesFrom>
</owl2xml:EquivalentClasses>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&MirrorObjects;powerStation"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:SubClassOf>
<owl2xml:Declaration>
<owl2xml:Class owl2xml:URI="&MirrorObjects;role"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
<owl2xml:Class owl2xml:URI="&MirrorObjects;scrape"/>
</owl2xml:Declaration>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&MirrorObjects;thermalPS"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects;powerStation"/>
</owl2xml:SubClassOf>
<owl2xml:Declaration>
<owl2xml:Class owl2xml:URI="&MirrorObjects;virtualAgent"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
<owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;affectsAsset"/>
</owl2xml:Declaration>
<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;aggregatedOf"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;aggregatedOf"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;composedOf"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;composedOf"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;contain"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;contain"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;firesInteraction"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;hasInteractionType"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;hasLoadedPS"/>
</owl2xml:Declaration>
<owl2xml:InverseObjectProperties>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;hasPowerStation"/>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;hasPresence"/>
</owl2xml:InverseObjectProperties>
<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;hasPowerStation"/>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;powerStation"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;involvesAsset"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;maintainsScape"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;ownsAsset"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;postsEvent"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;takesPartInInteraction"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;userPlaysRole"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;crtdBy"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;endDate"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;endTime"/>
</owl2xml:Declaration>

<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasAccessType"/>
</owl2xml:Declaration>

<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasContact"/>
</owl2xml:Declaration>

<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasCreatedate"/>
</owl2xml:Declaration>

<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasDescription"/>
</owl2xml:Declaration>

<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasGovernanceStyle"/>
</owl2xml:Declaration>

<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasName"/>
</owl2xml:Declaration>

<owl2xml:DataPropertyRange>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasPollutionIndex"/>
  <owl2xml:Datatype owl2xml:URI="&xsd;float"/>
</owl2xml:DataPropertyRange>

<owl2xml:Declaration>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasPriority"/>
</owl2xml:Declaration>

<owl2xml:EntityAnnotation>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasProductionAmps"/>
  <owl2xml:Annotation owl2xml:annotationURI="&rdfs;comment">
    <owl2xml:Constant>
      has high production amps
    </owl2xml:Constant>
  </owl2xml:Annotation>
</owl2xml:EntityAnnotation>
<owl2xml:DataProperty owl2xml:URI="&MirrorObjects;startDate"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
<owl2xml:DataProperty owl2xml:URI="&MirrorObjects;startTime"/>
</owl2xml:Declaration>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&MirrorObjects;presence"/>
<owl2xml:Individual owl2xml:URI="&MirrorObjects;chicago"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&MirrorObjects;chicago"/>
</owl2xml:ClassAssertion>
<owl2xml:DataPropertyAssertion>
<owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasPollutionIndex"/>
<owl2xml:Individual owl2xml:URI="&MirrorObjects;chicago"/>
<owl2xml:Constant owl2xml:datatypeURI="&xsd;float">1.5</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&MirrorObjects;presence"/>
<owl2xml:Individual owl2xml:URI="&MirrorObjects;columbus"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&MirrorObjects;columbus"/>
</owl2xml:ClassAssertion>
<owl2xml:DataPropertyAssertion>
<owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasPollutionIndex"/>
<owl2xml:Individual owl2xml:URI="&MirrorObjects;columbus"/>
<owl2xml:Constant owl2xml:datatypeURI="&xsd;float">0.96</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&MirrorObjects;presence"/>
</owl2xml:ClassAssertion>
<owl2xml:Individual owl2xml:URI="&MirrorObjects;delaware"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;delaware"/>
</owl2xml:ClassAssertion>
<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasPollutionIndex"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;delaware"/>
  <owl2xml:Constant owl2xml:datatypeURI="&xsd;float">0.78</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&MirrorObjects;powerStation"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;psA"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;psA"/>
</owl2xml:ClassAssertion>
<owl2xml:ObjectPropertyAssertion>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;hasPresence"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;psA"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;chicago"/>
</owl2xml:ObjectPropertyAssertion>
<owl2xml:ObjectPropertyAssertion>
  <owl2xml:ObjectProperty owl2xml:URI="&MirrorObjects;powersCity"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;psA"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;chicago"/>
</owl2xml:ObjectPropertyAssertion>
<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&MirrorObjects;hasProductionAmps"/>
  <owl2xml:Individual owl2xml:URI="&MirrorObjects;psA"/>
  <owl2xml:Constant owl2xml:datatypeURI="&xsd;float">2</owl2xml:Constant>
Mirror Core doesn’t have a domain mapping, since it is the underlying shared, generic ontology applicable to a wide range of domains. The individual Scapes derive or map their
view using this ontology to specialize it to a domain. Hence, only Scapes need to establish their domain understanding in the mirror framework.

- **Building the Service Ontology**

The service ontology for mirror core defines mirror services, and relates them using a set of relations. The following ontology is the actual service ontology, which uses the Service Relations ontology listed after it.

```xml
<?xml version="1.0"?>

<!DOCTYPE owl2xml:Ontology [ 
<!ENTITY owl "http://www.w3.org/2002/07/owl#" > 
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" > 
<!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" > 
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" > 
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" > 
<!ENTITY services "http://healthgame.cse.ohio-state.edu/2010/11/services.owl#" > 
<!ENTITY ServiceRelations "http://mirror.cse.ohio-state.edu/2010/11/ServiceRelations#" > 
]>}

<owl2xml:Ontology xmlns="http://healthgame.cse.ohio-state.edu/2010/11/services.owl#" 
xml:base="http://www.w3.org/2006/12/owl2-xml#" 
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" 
xmlns:services="http://healthgame.cse.ohio-state.edu/2010/11/services.owl#" 
xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#" 
xmlns:ServiceRelations="http://mirror.cse.ohio-state.edu/2010/11/ServiceRelations#" 
xmlns:owl="http://www.w3.org/2002/07/owl#" 
xmlns:xsd="http://www.w3.org/2001/XMLSchema#" 
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" 
owl2xml:URI="http://healthgame.cse.ohio-state.edu/2010/11/services.owl">

```
<owl2xml:Import>
  >http://mirror.cse.ohio-state.edu/2010/11/ServiceRelations</owl2xml:Import>
<owl2xml:Annotation owl2xml:annotationURI="&owl;versionInfo">
  <owl2xml:Constant owl2xml:datatypeURI="&rdf;PlainLiteral">1</owl2xml:Constant>
</owl2xml:Annotation>
<owl2xml:Import>
  >http://mirror.cse.ohio-state.edu/2010/11/ServiceRelations</owl2xml:Import>
<owl2xml:SubClassOf>
  <owl2xml:Class owl2xml:URI="&services;senseEnvironment"/>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;service"/>
</owl2xml:SubClassOf>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;Core"/>
  <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;FacebookLogin"/>
  <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;Login"/>
  <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;Remote"/>
  <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
</owl2xml:ClassAssertion>
<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;hasName"/>
  <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
  <owl2xml:Constant>
    >Facebook Login Service</owl2xml:Constant>
  </owl2xml:DataPropertyAssertion>
</owl2xml:DataPropertyAssertion>
<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;invokeStyle"/>
  <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
  <owl2xml:Constant>true</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
<owl2xml:Declaration>
  <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
</owl2xml:Declaration>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;GISEnabled"/>
  <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;GoogleMapsAPI"/>
  <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;Remote"/>
  <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ClassAssertion>
<owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
<owl2xml:Constant>true</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
<owl2xml:Declaration>
<owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:Declaration>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;GoogleSearch"/>
<owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;Remote"/>
<owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
</owl2xml:ClassAssertion>
<owl2xml:DataPropertyAssertion>
<owl2xml:DataProperty owl2xml:URI="&ServiceRelations;invokeStyle"/>
<owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
<owl2xml:Constant>http service</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
<owl2xml:DataPropertyAssertion>
<owl2xml:DataProperty owl2xml:URI="&ServiceRelations;isInvokable"/>
<owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
<owl2xml:Constant>true</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
<owl2xml:Declaration>
<owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
</owl2xml:Declaration>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&services;senseEnvironment"/>
<owl2xml:Individual owl2xml:URI="&services;getAlcoholInput"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&services;getAlcoholInput"/>
</owl2xml:ClassAssertion>
<owl2xml:Declaration>
<owl2xml:Individual owl2xml:URI="&services;getAlcoholInput"/>
</owl2xml:Declaration>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;service"/>
<owl2xml:Individual owl2xml:URI="&services;getNextGoal"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&services;getNextGoal"/>
</owl2xml:ClassAssertion>
<owl2xml:Declaration>
<owl2xml:Individual owl2xml:URI="&services;getNextGoal"/>
</owl2xml:Declaration>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ImportExport"/>
<owl2xml:Individual owl2xml:URI="&services;importExport"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;Ontology"/>
<owl2xml:Individual owl2xml:URI="&services;importExport"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&services;importExport"/>
</owl2xml:ClassAssertion>
<owl2xml:Declaration>
<owl2xml:Individual owl2xml:URI="&services;importExport"/>
</owl2xml:Declaration>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;DataAcquisition"/>
<owl2xml:Individual owl2xml:URI="&services;importExportSchemaData"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ImportExport"/>
<owl2xml:Individual owl2xml:URI="&services;importExportSchemaData"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;SchemaAcquisition"/>
<owl2xml:Individual owl2xml:URI="&services;importExportSchemaData"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&services;importExportSchemaData"/>
</owl2xml:ClassAssertion>
<owl2xml:Declaration>
<owl2xml:Individual owl2xml:URI="&services;importExportSchemaData"/>
</owl2xml:Declaration>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;Login"/>
<owl2xml:Individual owl2xml:URI="&services;login"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&ServiceRelations;service"/>
<owl2xml:Individual owl2xml:URI="&services;login"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&services;login"/>
</owl2xml:ClassAssertion>
<owl2xml:Ontology xmlns="http://mirror.cse.ohio-state.edu/2010/11/ServiceRelations#" xml:base="http://www.w3.org/2006/12/owl2-xml#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
xmlns:serviceRelations="http://mirror.cse.ohio-state.edu/2010/11/ServiceRelations#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
owl2xml:URI="http://mirror.cse.ohio-state.edu/2010/11/ServiceRelations">
<owl2xml:Annotation owl2xml:annotationURI="&owl;versionInfo">
<owl2xml:Constant owl2xml:datatypeURI="&rdf;PlainLiteral">1</owl2xml:Constant>
</owl2xml:Annotation>
<owl2xml:Annotation owl2xml:annotationURI="&rdfs;label">
<owl2xml:Constant owl2xml:datatypeURI="&rdf;PlainLiteral">Global Service Relations</owl2xml:Constant>
</owl2xml:Annotation>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&serviceRelations;CRUD"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&serviceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&serviceRelations;Core"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&serviceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&ServiceRelations;Ontology"/>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&ServiceRelations;Remote"/>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ScapeDevelopment"/>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&ServiceRelations;SchemaAcquisition"/>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&ServiceRelations;Search"/>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&ServiceRelations;Shared"/>
<owl2xml:Class owl2xml:URI="&ServiceRelations;ServiceTypes"/>
</owl2xml:SubClassOf>
<owl2xml:Declaration>
<owl2xml:Class owl2xml:URI="&ServiceRelations;anyOrder"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
<owl2xml:Class owl2xml:URI="&ServiceRelations;atomicProcess"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
<owl2xml:Declaration/>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;hasInput"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;hasOutput"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;hasParameter"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;hasParticipant"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;hasPostcondition"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;hasPrecondition"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;hasProcesses"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;hasResult"/>
</owl2xml:Declaration>
<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;imports"/>
</owl2xml:Declaration>
<owl2xml:InverseObjectProperties>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;realizedBy"/>
  <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;realizes"/>
</owl2xml:InverseObjectProperties>
<owl2xml:Declaration>
Building the Mirror-Upper Domain Ontology

This component is an upper ontology of the high level goals modeled by the domains known within the Mirror Framework. It describes what part of the actual real world the cyberinfrastructure currently knows. Individual Scapes that enable these individual partial realities let themselves known by augmenting to this ontology. This ontology is derived from general upper ontologies like CYC, or SWEET.
<owl2xml:SubClassOf>
  <owl2xml:Class owl2xml:URI="&MOG;HealthManagement"/>
  <owl2xml:Class owl2xml:URI="&MOG;Living"/>
</owl2xml:SubClassOf>

<owl2xml:EntityAnnotation>
  <owl2xml:Class owl2xml:URI="&MOG;HealthManagement"/>
  <owl2xml:Annotation owl2xml:annotationURI="&rdfs;comment">
    <owl2xml:Constant>
      Healthy living suggestion
    </owl2xml:Constant>
  </owl2xml:Annotation>
</owl2xml:EntityAnnotation>

<owl2xml:EntityAnnotation>
  <owl2xml:Class owl2xml:URI="&MOG;HealthManagement"/>
  <owl2xml:Annotation owl2xml:annotationURI="&rdfs;label">
    <owl2xml:Constant>
      Indicates a suggestion service for healthy living
    </owl2xml:Constant>
  </owl2xml:Annotation>
</owl2xml:EntityAnnotation>

<owl2xml:EntityAnnotation>
  <owl2xml:Class owl2xml:URI="&MOG;Living"/>
  <owl2xml:Annotation owl2xml:annotationURI="&rdfs;comment">
    <owl2xml:Constant>
    </owl2xml:Constant>
  </owl2xml:Annotation>
</owl2xml:EntityAnnotation>

<owl2xml:SubClassOf>
  <owl2xml:Class owl2xml:URI="&MOG;PhysicalGaming"/>
  <owl2xml:Class owl2xml:URI="&MOG;Gaming"/>
</owl2xml:SubClassOf>
• Building the API Configuration for Core

Ultimately, we consolidate all this object and behavior description into the Scape configuration file, along with any extraneous definitions and descriptions of objects, services that may be chosen to describe separately. This also lists the various rules in different formats; however, we defer the processing of externally modeled rules to future work following the reference implementation for the Core and Scape.

BEGIN

// This is a comment line
// Include all the ontologies using the following link commands:
//MOG: Upper Domain Ontology, OOG: Upper Dynamics Ontology, SOG: Service Ontology, WSDL: the deployed web service definition and description
@MOG <http://mirror.cse.ohio-state.edu/2010/11/MOG#>
//No DMOG!
//##Services defines a section for services
##Services

   <MirrorServiceRelations:service rdf:ID="exportOntology">
      <rdf:type rdf:resource="&MirrorServiceDefinitions;OntologyServices"/>
      <rdf:type rdf:resource="&MirrorServiceDefinitions;ImportExportServices"/>
      <rdf:type rdf:resource="&MirrorServiceDefinitions;SharedServices"/>
   </MirrorServiceRelations:service>

   <MirrorServiceRelations:service rdf:ID="exportInstances">
      <rdf:type rdf:resource="&MirrorServiceDefinitions;DataAcquisitionServices"/>
      <rdf:type rdf:resource="&MirrorServiceDefinitions;ImportExportServices"/>
      <rdf:type rdf:resource="&MirrorServiceDefinitions;SharedServices"/>
   </MirrorServiceRelations:service>

   <MirrorServiceRelations:service rdf:ID="exportSchema">
      <rdf:type rdf:resource="&MirrorServiceDefinitions;SchemaAcquisitionServices"/>
      <rdf:type rdf:resource="&MirrorServiceDefinitions;ImportExportServices"/>
      <rdf:type rdf:resource="&MirrorServiceDefinitions;SharedServices"/>
   </MirrorServiceRelations:service>

   <MirrorServiceRelations:service rdf:ID="updateInstances">
<rdf:type rdf:resource="&MirrorServiceDefinitions;DataAcquisitionServices"/>

<rdf:type rdf:resource="&MirrorServiceDefinitions;SharedServices"/>

</MirrorServiceRelations:service>

//##Rules

//Rules section

//#includeJessRules<JessRulesFile.jess>  //Jess rules to be served to Jess engine

//#includeETPRules<ETPRulesFile.etp>     //describes event triggered procedures

//These above rules in different formats specify what behavior they exhibit with the use of what objects – all only in regard to the current Scape objects. Note, however, that the current Scape objects could, nonetheless, be derived from either Mirror’s objects, current Scape’s other objects, or another Scapes objects as described in the ontologies.

END

4.6.2 Implemented Core Services

Import/Export/Search services

A Scape at its basic configuration is a set of services built on some assets in mirror. The idea is to let this whole system evolve in a flexible way. Therefore, we envision the Scape configuration as a markup file specifying the objects/assets it wants to use, the services around that assets that it wants to use, and the rules that it will enforce on the local copy of those assets – the rules help identify the behavioral aspects of a Scape. Once this configuration is ready, the Scape developer either “includes” services from already such developed Scapes, or develops and exposes new services, so that Scapes developed later can use these services in their configuration.
Using this methodology, a Scape being built is able to import services from other Scapes in the system. However, it is up to the individual Scape to then manage the consistency concerns with other Scape objects. The import/export services are provided on the Scape configuration file, the ontological specifications, Scape or Mirror’s database schema(s), and actual data as well. Using the Scape configuration file, the configuration of a Scape can be exported in a markup text format (same as the configuration file) that lists the Core and Scape specific entities used, services used and provided and rules that are executed as a part of this Scape – rules referenced from mirror core as well as rules specified to be executed in Scape. These services also include import/export of relational object schema and the actual instance data for population of a relational schema. Following figure shows the service implementation in the framework.
Above figure shows serialized instances being exported in a universally (in Java World) understandable format. This data can be directly loaded into any Java program for processing, and onto other platforms like .NET with necessary transformations. The conversions and interoperability between different platforms is out of scope of this thesis.

The Core also provides search services through query language on the ontology specification to let the Scape developer locate the relevant Scapes, services and modeling done in the system. This is currently externally done for the demonstrative purposes of the validation in this thesis, though the reasoners and tools required for this are actually integrated in the existing Mirror framework. This is presented and discussed more in Chapter 6.

**Mirror-Scape publish - subscribe and Mediation across Scapes**

Using this distributed service model entails a way to transfer the required data and control flow across all the service providers and users. Mediation of this control and data flow is
managed through use of an Enterprise Service Bus. The objects or entities in Mirror that Scapes would want to manage or change are specified in their configuration file. Naturally, this creates redundancy in data and spawns a variety of consistency and coherency problems related to the data. To address this, currently we use publish-subscribe model that gives a Scape complete ownership of the data specified in the configuration file. However, this data is not the data that belongs to Mirror; it is copy in the Scape’s local context and is managed in the Scape’s operational environment. The reason for keeping this model is having consistency across Scapes if the Scapes wish to do so, but the underlying Core is always consistent with the real world. Any update to data definition in Mirror will outdate the Scapes that are already operational with a copy of Mirror data with old data definition. However, it is up to the discretion of Scape developer to keep the old copy or take the effort to keep it up to date.

Figure 16: A Data Sharing and Mediation Service Implementation
The SOAP based web service definition for this import/export services is shown below. Scapes use these deployed web services to share their data with Mirror Core.

![XML code](image)

**Figure 17**: A Data Sharing & Mediation Service Definition
Figure 18 demonstrates how the sharing and re-use, in terms of objects, components, services, knowledge and processes, is enabled by the framework.

Core Entity services

The core mirror services include Create, Review, Update, Delete (CRUD) operations around objects in Mirror Core. These data objects include, for example, assets, users, roles, Scapes, groups, etc. Some of these services are exposed to only administrative user interface provided on the Core, and all of them are not exposed as web services because the validation does not have any Scape or user using this functionality remotely. However, exposing them as web services in the framework is easily possible by just adding some annotations in the implementation of the framework.
Following figure shows the implemented core entity objects, the next figure shows the front end component that enables the core entity services around these objects.

Figure 19: Implemented Core Objects & Services at the backend (Model in MVC)
Figure 20: The front end (View & Controller in MVC) for Core Entity Services
Figure 21: Administrative UI (View in MVC) to the Core Services

Mirror runtime services

The key idea of Mirror is to capture the run-time behavior of the Scapes and itself to analyze and identify possible areas of feature and performance enhancement. The Mirror cyberinfrastructure logs all the user interactions through roles and transactions. These interactions can be specific to a Scape or with Mirror itself. Analyzing these interactions will help uncover user behavioral trends. The (low level software/firmware) system operations and events can also be easily logged in Mirror through the integrated libraries; but logging system operations and events is not enabled in the current implementation. Following figure shows the libraries used for such logging.
The runtime system also includes the business rules that are executed as a result of Scape configurations and the Mirror core. The rule sets can be totally disjoint, or can be overlapping in terms of assets, users, interactions and roles they include. They can also be executing at different places if they are totally disjoint and the Mirror core doesn’t need to worry about Scape rules. If however, they are overlapping, the Mirror rules will assume priority over conflicting Scape configuration rules. Following figure shows the implementation of the rule engine in Mirror Core. We demonstrate the usage of such engine in Chapter 6.
Role-based security services

Mirror supports role-based access to its data, operations and services through a governance mechanism. A snapshot of a subset of this governance mechanism is shown in User Stories section in Appendix A.
Following services are NOT implemented in the current prototype of Mirror Core:

Visualization, GIS***** layer & Location Based services

Each asset in the Mirror core as well as Scapes is associated with a geographical position in spatial dimension and a value (which may or may not be relevant in all contexts) in time dimension. These attributes are represented using a ‘presence’ relation, and contains all the information that will help detect the position, orientation, speed (if applicable) of the object. This helps us in enabling GIS services and providing user specific services based on live inputs from various interfaced positioning devices around the globe. Apart from static usage, this also enables simulations and visualizations on top of static GIS information to give the users different views of the same underlying data.

Different end user devices can act as clients or agents in a Scape developed. Such devices with GPS††† ability can help further in reaching out with the best services to the user in faster ways. An example of such a service could be suggestions based on current location of a user – suggesting recreation/hotel/restaurant/hospitals around them, suggesting shortest path to the next user goal through use of GIS systems and/or interfacing with Google Maps.

Monitoring dashboard services

Mirror services can be extended to include monitoring and logging services that can hook on to different places in the software components so as to capture critical operational and performance data and display it using an intuitive dashboard for performance analytics and improvement.

Tools for analysis and decision-making

The idea of using distributed services lets us build on top of Scapes to eventually build an evolving, bigger and comprehensive system. Ultimately, the ideal Mirror system will include all interactions that occur in a real world embedded in their respective contexts, but

***** Geographical Information Systems

††††† Global Positioning System
noticeable to other components. This will help simulate the real world actions with fair rules and laws as in the real world, and thus aid decision making through use of various analysis tools. One example is the ODJFS (Ohio Department of Job and Family Services) Scape implemented in the CETI that uses external CSR (Customer Service Requests) data to feed into an (external) analysis tool to figure out different CSR characteristics across different sub-organizations in an enterprise.

### 4.7 Scape Configuration & Development Process

Assuming the definitions and concepts introduced earlier, this section describes the development of a Scape application using the Mirror framework. A Scape is derived using the following work products:

- a Scape API using the syntax discussed below
- a set of ontology specifications described earlier

**Figure 24: Scape Development Process**
Following are the sequence of steps to build and describe a Scape in the Mirror system. Note that all the Mirror Core specifications are already established before developing a Scape with the framework.

- Identify objects in the Scape, build the Domain ontology
  - Derived from the Object Interaction diagram and domain model

- Design the Scape, build a service definition file, link the web service definitions if applicable, and build the service ontology.
  - Derived from use-cases, design, services identified

- Build the Domain-Mapping Ontology
  - Derived from domain modeling, domain understanding and expertise

- Describe the Scape in real-world, i.e. build a Mirror Upper Domain Ontology for the Scape
  - Derived from generic upper ontologies and Mirror Upper Domain Ontology

- Build the Scape API file

At this stage, an understanding of some of the Scape’s static and behavioral aspects is incorporated in the system. In this prototype Mirror implementation, manual intervention is required for Scape development, which later would be achieved in a semi-automated manner depicted in the figure below. In short, the user queries all the ontology specifications and configurations to dynamically compose a new Scape API with objects, rules, services from different Scapes. This Scape API is parsed and fed to a tool to extract the referenced objects, dynamics (rules), and services using the ontology specifications. All this information is combined and fed to another set of tools that aid ontology to code conversions (as mentioned in section 1.3). This is how the Mirror framework can support a CASE‡‡‡‡‡‡‡‡ type

‡‡‡‡‡‡‡‡ CASE: Computer Aided Software Engineering
feature and a crucial Service Discovery, Configuration and Composition ability. The implementation of these features is left to the future work, but directions and leads for this work are set through the background research identified in section 1.3.

Figure 25: Scape Development Semi-Automation
5. Implemented Reference Architecture: Scapes

Using the process described in section 4.7, this chapter presents the implementations of the Scapes (sections 3.2-3.4), with the Mirror framework. These implementations are then used to validate the operation of the framework, as specified by sections 2.3 and 3.6.

5.1 HealthGame Scape

We now describe the process for developing HealthGame specifications and implementing the Scape with Mirror Framework.

- Identifying HealthGame objects, building Domain Ontology

The interfacing of UI components and other “non-model implementation” classes are not of much interest to the Mirror System. The model classes that the Scape wants Mirror Core to be aware of are modeled in the domain ontology for the Mirror framework. For example, HealthGame Scape has the following objects

User – E.g. John, Alice

Interaction – E.g. sendTask, getResponse, recordInteraction, etc

Goal – Call friend, walk home, exercise in gym, drink alcohol, call cab, etc

Event – User Inputs, GIS based location updates through devices, etc (The Sensing component)
The object interaction model for HealthGame is shown below.

![Diagram of HealthGame Object Interaction/Domain Model](image)

**Figure 26: HealthGame Object Interaction/Domain Model**

- **Building Domain Mapping Ontology for HealthGame**

  Of the objects that HealthGame OOG describes, the ones that HealthGame inherits from Mirror system are User and Interaction, for example.

  Following is the domain and the domain mapping ontology for HealthGame.

  ```xml
  <?xml version="1.0"?>
  
  <!DOCTYPE owl2xml:Ontology [ 
  ```
  ```
  ```
  ```
  ```
  ```
  ```
  ![DOCTYPE owl2xml:Ontology [
<owl2xml:Class owl2xml:URI="&objects;agent"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects;mirrorUser"/>
</owl2xml:EquivalentClasses>
<owl2xml:EntityAnnotation>
<owl2xml:Class owl2xml:URI="&objects;agent"/>
<owl2xml:Annotation owl2xml:annotationURI="&rdfs;label">
<owl2xml:Constant>Agent</owl2xml:Constant>
</owl2xml:Annotation>
</owl2xml:EntityAnnotation>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&objects;campusDiningPlaces"/>
<owl2xml:Class owl2xml:URI="&objects;places"/>
</owl2xml:SubClassOf>
<owl2xml:EquivalentClasses>
<owl2xml:Class owl2xml:URI="&objects;drunkAgents"/>
<owl2xml:DataMinCardinality owl2xml:cardinality="1">
<owl2xml:DataProperty owl2xml:URI="&objects;alcoholLevel"/>
</owl2xml:DataMinCardinality>
</owl2xml:EquivalentClasses>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&objects;finalizeTownRequest"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects;event"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&objects;getResponse"/>
<owl2xml:Class owl2xml:URI="&objects;interaction"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&objects;gym"/>
<owl2xml:Class owl2xml:URI="&objects;places"/>
</owl2xml:SubClassOf>
<owl2xml:EquivalentClasses>
<owl2xml:Class owl2xml:URI="&objects;interaction"/>
<owl2xml:EquivalentClasses><owl2xml:Class owl2xml:URI="&MirrorObjects;interactionType"/></owl2xml:EquivalentClasses>
<owl2xml:EquivalentClasses>
<owl2xml:Class owl2xml:URI="&objects;interactionexecrecord"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects;interaction"/>
</owl2xml:EquivalentClasses>
<owl2xml:EquivalentClasses>
<owl2xml:Class owl2xml:URI="&objects;ownedPlaces"/>
<owl2xml:ObjectSomeValuesFrom>
<owl2xml:ObjectProperty owl2xml:URI="&objects;ownedBy"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects;mirrorUser"/>
</owl2xml:ObjectSomeValuesFrom>
</owl2xml:EquivalentClasses>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&objects;ownedPlaces"/>
<owl2xml:Class owl2xml:URI="&objects;places"/>
</owl2xml:SubClassOf>
<owl2xml:EntityAnnotation>
<owl2xml:Class owl2xml:URI="&objects;ownedPlaces"/>
<owl2xml:Annotation owl2xml:annotationURI="&rdfs;comment">
<owl2xml:Constant>privately owned places</owl2xml:Constant>
</owl2xml:Annotation>
</owl2xml:EntityAnnotation>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&objects;ownedPlaces"/>
<owl2xml:Class owl2xml:URI="&MirrorObjects;asset"/>
</owl2xml:SubClassOf>
<owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&objects;publishToFacebook"/>
<owl2xml:Class owl2xml:URI="&objects;interaction"/>
</owl2xml:SubClassOf>
<owl2xml:EquivalentClasses>
<owl2xml:Class owl2xml:URI="&objects;agent"/>

<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;chosenGoal"/>
  <owl2xml:Class owl2xml:URI="&objects;goal"/>
</owl2xml:ObjectPropertyDomain>

<owl2xml:Declaration>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;goalIsInScenario"/>
</owl2xml:Declaration>

<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;hasGoal"/>
  <owl2xml:Class owl2xml:URI="&objects;goal"/>
</owl2xml:ObjectPropertyRange>

<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;hasScenario"/>
  <owl2xml:Class owl2xml:URI="&objects;scenario"/>
</owl2xml:ObjectPropertyRange>

<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;isFormedBy"/>
  <owl2xml:Class owl2xml:URI="&objects;interactionexecrecord"/>
</owl2xml:ObjectPropertyDomain>

<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;isFormedBy"/>
  <owl2xml:Class owl2xml:URI="&objects;interaction"/>
</owl2xml:ObjectPropertyRange>

<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;isInteractionOfType"/>
  <owl2xml:Class owl2xml:URI="&objects;interactionexecrecord"/>
</owl2xml:ObjectPropertyDomain>

<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;isInteractionOfType"/>
  <owl2xml:Class owl2xml:URI="&objects;interaction"/>
</owl2xml:ObjectPropertyRange>
<owlxml:ObjectPropertyDomain>
  <owlxml:ObjectProperty owlxml:URI="&objects;leadsTo"/>
  <owlxml:Class owlxml:URI="&objects;interactionexecrecord"/>
</owlxml:ObjectPropertyDomain>

<owlxml:ObjectPropertyRange>
  <owlxml:ObjectProperty owlxml:URI="&objects;leadsTo"/>
  <owlxml:Class owlxml:URI="&objects;goal"/>
</owlxml:ObjectPropertyRange>

<owlxml:InverseObjectProperties>
  <owlxml:ObjectProperty owlxml:URI="&objects;ownedBy"/>
  <owlxml:ObjectProperty owlxml:URI="&objects;purchasesGym"/>
</owlxml:InverseObjectProperties>

<owlxml:EquivalentObjectProperties>
  <owlxml:ObjectProperty owlxml:URI="&objects;purchasesGym"/>
  <owlxml:ObjectProperty owlxml:URI="&MirrorObjects;ownsAsset"/>
</owlxml:EquivalentObjectProperties>

<owlxml:ObjectPropertyDomain>
  <owlxml:ObjectProperty owlxml:URI="&objects;scenarioHasGoal"/>
  <owlxml:Class owlxml:URI="&objects;scenario"/>
</owlxml:ObjectPropertyDomain>

<owlxml:ObjectPropertyRange>
  <owlxml:ObjectProperty owlxml:URI="&objects;scenarioHasGoal"/>
  <owlxml:Class owlxml:URI="&objects;interaction"/>
</owlxml:ObjectPropertyRange>

<owlxml:EquivalentObjectProperties>
  <owlxml:ObjectProperty owlxml:URI="&objects;submitsFinalizeRequest"/>
  <owlxml:ObjectProperty owlxml:URI="&MirrorObjects;postsEvent"/>
</owlxml:EquivalentObjectProperties>

<owlxml:ObjectPropertyDomain>
  <owlxml:ObjectProperty owlxml:URI="&objects;triggers"/>
  <owlxml:Class owlxml:URI="&objects;agent"/>
</owlxml:ObjectPropertyDomain>

<owlxml:ObjectPropertyRange>
<owl2xml:ObjectProperty owl2xml:URI="&objects;triggers"/>
<owl2xml:Class owl2xml:URI="&objects;interaction"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:DataPropertyDomain>
<owl2xml:DataProperty owl2xml:URI="&objects;alcoholLevel"/>
<owl2xml:Class owl2xml:URI="&objects;agent"/>
</owl2xml:DataPropertyDomain>
<owl2xml:DataPropertyRange>
<owl2xml:DataProperty owl2xml:URI="&objects;alcoholLevel"/>
<owl2xml:Datatype owl2xml:URI="&xsd;float"/>
</owl2xml:DataPropertyRange>
<owl2xml:DataPropertyDomain>
<owl2xml:DataProperty owl2xml:URI="&objects;hasId"/>
<owl2xml:Class owl2xml:URI="&objects;agent"/>
</owl2xml:DataPropertyDomain>
<owl2xml:DataPropertyRange>
<owl2xml:DataProperty owl2xml:URI="&objects;hasId"/>
<owl2xml:Datatype owl2xml:URI="&xsd;integer"/>
</owl2xml:DataPropertyRange>
<owl2xml:DataPropertyDomain>
<owl2xml:DataProperty owl2xml:URI="&objects;hasName"/>
<owl2xml:Class owl2xml:URI="&objects;agent"/>
</owl2xml:DataPropertyDomain>
<owl2xml:DataPropertyRange>
<owl2xml:DataProperty owl2xml:URI="&objects;hasName"/>
<owl2xml:Datatype owl2xml:URI="&xsd;string"/>
</owl2xml:DataPropertyRange>
<owl2xml:Declaration>
<owl2xml:DataProperty owl2xml:URI="&objects;hasScenarioId"/>
</owl2xml:Declaration>
<owl2xml:DataPropertyDomain>
<owl2xml:DataProperty owl2xml:URI="&objects;hasScore"/>
<owl2xml:Class owl2xml:URI="&objects;agent"/>
</owl2xml:DataPropertyRange>
<owl2xml:DataProperty owl2xml:URI="&objects;hasScore"/>
<owl2xml:Datatype owl2xml:URI="&xsd;integer"/>
</owl2xml:DataPropertyRange>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&objects;campusDiningPlaces"/>
<owl2xml:Individual owl2xml:URI="&objects;KennedyCommons"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&objects;KennedyCommons"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&objects;campusDiningPlaces"/>
<owl2xml:Individual owl2xml:URI="&objects;WoodyHayesTavern"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&objects;WoodyHayesTavern"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&objects;goal"/>
<owl2xml:Individual owl2xml:URI="&objects;callCab"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&objects;callCab"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&objects;goal"/>
<owl2xml:Individual owl2xml:URI="&objects;callFriend"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&objects;callFriend"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&objects;agent"/>
  <owl2xml:Individual owl2xml:URI="&objects;ceti"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&objects;ceti"/>
</owl2xml:ClassAssertion>

<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&objects;alcoholLevel"/>
  <owl2xml:Individual owl2xml:URI="&objects;ceti"/>
  <owl2xml:Constant owl2xml:datatypeURI="&xsd;float">0.0</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&objects;hasName"/>
  <owl2xml:Individual owl2xml:URI="&objects;ceti"/>
  <owl2xml:Constant owl2xml:datatypeURI="&xsd;string">ceti</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&objects;goal"/>
  <owl2xml:Individual owl2xml:URI="&objects;driveHome"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&objects;driveHome"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&objects;finalizeTownRequest"/>
  <owl2xml:Individual owl2xml:URI="&objects;finReq1"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&objects;finReq1"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&objects;gym"/>
<owl2xml:Individual owl2xml:URI="&objects;jos"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&objects;newGym"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&objects;gym"/>
<owl2xml:Individual owl2xml:URI="&objects;newGym"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&objects;newGym"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&objects;gym"/>
<owl2xml:Individual owl2xml:URI="&objects;rpac"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&owl;Thing"/>
<owl2xml:Individual owl2xml:URI="&objects;rpac"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
<owl2xml:Class owl2xml:URI="&objects;agent"/>
<owl2xml:Individual owl2xml:URI="&objects;user123"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&objects;user123"/>
</owl2xml:ClassAssertion>

<owl2xml:ObjectPropertyAssertion>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;chosenGoal"/>
  <owl2xml:Individual owl2xml:URI="&objects;user123"/>
  <owl2xml:Individual owl2xml:URI="&objects;driveHome"/>
</owl2xml:ObjectPropertyAssertion>

<owl2xml:ObjectPropertyAssertion>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;purchasesGym"/>
  <owl2xml:Individual owl2xml:URI="&objects;user123"/>
  <owl2xml:Individual owl2xml:URI="&objects;newGym"/>
</owl2xml:ObjectPropertyAssertion>

<owl2xml:ObjectPropertyAssertion>
  <owl2xml:ObjectProperty owl2xml:URI="&objects;submitsFinalizeRequest"/>
  <owl2xml:Individual owl2xml:URI="&objects;user123"/>
  <owl2xml:Individual owl2xml:URI="&objects;finReq1"/>
</owl2xml:ObjectPropertyAssertion>

<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&objects;alcoholLevel"/>
  <owl2xml:Individual owl2xml:URI="&objects;user123"/>
  <owl2xml:Constant owl2xml:datatypeURI="&xsd;float">1.8</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&objects;hasId"/>
  <owl2xml:Individual owl2xml:URI="&objects;user123"/>
  <owl2xml:Constant owl2xml:datatypeURI="&xsd;integer">1</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&objects;hasName"/>
  <owl2xml:Individual owl2xml:URI="&objects;user123"/>
</owl2xml:DataPropertyAssertion>
Building the Service Ontology

The service ontology lists and relates HealthGame services with each other using a predefined set of relations stored in Core.
<owl2xml:ClassAssertion>
    <owl2xml:Class owl2xml:URI="&ServiceRelations;Remote"/>
    <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
</owl2xml:ClassAssertion>

<owl2xml:DataPropertyAssertion>
    <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;hasName"/>
    <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
    <owl2xml:Constant>
        Facebook Login Service
    </owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:DataPropertyAssertion>
    <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;invokeStyle"/>
    <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
    <owl2xml:Constant>
        imported package
    </owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:DataPropertyAssertion>
    <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;isInvokable"/>
    <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
    <owl2xml:Constant>true
</owl2xml:DataPropertyAssertion>

<owl2xml:Declaration>
    <owl2xml:Individual owl2xml:URI="&services;FBLogin"/>
</owl2xml:Declaration>

<owl2xml:ClassAssertion>
    <owl2xml:Class owl2xml:URI="&ServiceRelations;GISEnabled"/>
    <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
    <owl2xml:Class owl2xml:URI="&ServiceRelations;GoogleMapsAPI"/>
    <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ClassAssertion>
<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;Remote"/>
  <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ClassAssertion>

<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;isInvokable"/>
  <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
  <owl2xml:Constant>true</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:Declaration>
  <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:Declaration>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;GoogleSearch"/>
  <owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;Remote"/>
  <owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
</owl2xml:ClassAssertion>

<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;invokeStyle"/>
  <owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
  <owl2xml:Constant>http service</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;isInvokable"/>
  <owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
  <owl2xml:Constant>true</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:Declaration>
  <owl2xml:Individual owl2xml:URI="&services;GSearchAPI"/>
</owl2xml:Declaration>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&services;senseEnvironment"/>
  <owl2xml:Individual owl2xml:URI="&services;getAlcoholInput"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&services;getAlcoholInput"/>
</owl2xml:ClassAssertion>

<owl2xml:Declaration>
  <owl2xml:Individual owl2xml:URI="&services;getAlcoholInput"/>
</owl2xml:Declaration>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;service"/>
  <owl2xml:Individual owl2xml:URI="&services;getNextGoal"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  <owl2xml:Individual owl2xml:URI="&services;getNextGoal"/>
</owl2xml:ClassAssertion>

<owl2xml:Declaration>
  <owl2xml:Individual owl2xml:URI="&services;getNextGoal"/>
</owl2xml:Declaration>

<owl2xml:ClassAssertion>
  <owl2xml:Class owl2xml:URI="&ServiceRelations;ImportExport"/>
  <owl2xml:Individual owl2xml:URI="&services;importExport"/>
</owl2xml:ClassAssertion>
<owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ObjectPropertyAssertion>

<owl2xml:DataPropertyAssertion>
    <owl2xml:DataProperty owl2xml:URI="&ServiceRelations;hasName"/>
    <owl2xml:Individual owl2xml:URI="&services;putMarkers"/>
    <owl2xml:Constant>Google Maps</owl2xml:Constant>
</owl2xml:DataPropertyAssertion>

<owl2xml:Declaration>
    <owl2xml:Individual owl2xml:URI="&services;putMarkers"/>
</owl2xml:Declaration>

<owl2xml:ClassAssertion>
    <owl2xml:Class owl2xml:URI="&ServiceRelations;GISEnabled"/>
    <owl2xml:Individual owl2xml:URI="&services;switchView"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
    <owl2xml:Class owl2xml:URI="&ServiceRelations;service"/>
    <owl2xml:Individual owl2xml:URI="&services;switchView"/>
</owl2xml:ClassAssertion>

<owl2xml:ClassAssertion>
    <owl2xml:Class owl2xml:URI="&owl;Thing"/>
    <owl2xml:Individual owl2xml:URI="&services;switchView"/>
</owl2xml:ClassAssertion>

<owl2xml:ObjectPropertyAssertion>
    <owl2xml:ObjectProperty owl2xml:URI="&ServiceRelations;enabledBy"/>
    <owl2xml:Individual owl2xml:URI="&services;switchView"/>
    <owl2xml:Individual owl2xml:URI="&services;GMapsAPI"/>
</owl2xml:ObjectPropertyAssertion>

<owl2xml:Declaration>
    <owl2xml:Individual owl2xml:URI="&services;switchView"/>
</owl2xml:Declaration>

</owl2xml:Ontology>

<!-- Generated by the OWL API (version 2.2.1.1138) http://owlapi.sourceforge.net -->
• Relating to the Mirror “Upper” Domain Ontology

Having an understanding of the objects and relations at intra-Scape level, we need to also relate the HealthGame Scape (software application) with respect to other Scapes modeled in the system. This is done by relating the HealthGame scape to the Mirror Upper Domain ontology.

```xml
<?xml version="1.0"?>

<!DOCTYPE Ontology [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/rdf-syntax-ns#" >
  <!ENTITY MOG "http://mirror.cse.ohio-state.edu/2010/11/MOG#" >
  <!ENTITY MirrorObjects "http://mirror.cse.ohio-state.edu/2010/11/objects#" >
  <!ENTITY base "http://mirror.cse.ohio-state.edu/2010/11/MOGS/HealthGame" >
  <!ENTITY hg-mog "file:/C:/Users/test/Tools/misc/healthgamedatafiles/ontology/hg-mog.owl#" >
]>

<Ontology xmlns="http://www.w3.org/2006/12/owl2-xml#"
    xml:base="http://www.w3.org/2006/12/owl2-xml#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:hg-mog="file:/C:/Users/test/Tools/misc/healthgamedatafiles/ontology/hg-mog.owl#"
    xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
    xmlns:MirrorObjects="http://mirror.cse.ohio-state.edu/2010/11/objects#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"/>
```
Building The Scape API for HealthGame

As the final step, we consolidate all this information into a Scape API file, along with further information describing the behavior that HealthGame wants Mirror framework to be aware of.

BEGIN

// This is a comment line
// include the Ontology specifications

@MOG <http://mirror.cse.ohio-state.edu/2010/11/MOG/HealthGame#>
@DMOG <http://mirror.cse.ohio-state.edu/2010/11/DMOG/HealthGame#>
@SOG <http://mirror.cse.ohio-state.edu/2010/11/HealthGameServiceOntology#>
@OOG <http://healthgame.cse.ohio-state.edu/2010/11/objects#>
@WSDL <http://mirror.cse.ohio-state.edu/mirror-mirror-ejb/DataPusherService?wsdl>

//##Services defines a section for services
##Services

    <MirrorServiceRelations:service rdf:ID="exportOntology">
        <rdf:type rdf:resource="&MirrorServiceDefinitions;OntologyServices"/>
        <rdf:type rdf:resource="&MirrorServiceDefinitions;ImportExportServices"/>
        <rdf:type rdf:resource="&MirrorServiceDefinitions;SharedServices"/>
    </MirrorServiceRelations:service>

    <MirrorServiceRelations:service rdf:ID="exportInstances">
        <rdf:type rdf:resource="&MirrorServiceDefinitions;DataAcquisitionServices"/>
        <rdf:type rdf:resource="&MirrorServiceDefinitions;ImportExportServices"/>
        <rdf:type rdf:resource="&MirrorServiceDefinitions;SharedServices"/>
    </MirrorServiceRelations:service>

    <MirrorServiceRelations:service rdf:ID="exportSchema">
        <rdf:type rdf:resource="&MirrorServiceDefinitions;SchemaAcquisitionServices"/>
        <rdf:type rdf:resource="&MirrorServiceDefinitions;ImportExportServices"/>
    </MirrorServiceRelations:service>
<rdf:type rdf:resource="&MirrorServiceDefinitions;SharedServices"/>
</MirrorServiceRelations:service>

<MirrorServiceRelations:service rdf:ID="updateInstances">
  <rdf:type rdf:resource="&MirrorServiceDefinitions;DataAcquisitionServices"/>
  <rdf:type rdf:resource="&MirrorServiceDefinitions;SharedServices"/>
</MirrorServiceRelations:service>

## Rules

//Rules section

// #includeJessRules<[JessRulesFile.jess]> // Jess rules to be served to Jess engine
// #includeETPRules<[ETPRulesFile.etp]> // describes event triggered procedures

// These above rules in different formats specify what behavior they exhibit with the use of what objects – all only in regard to the current scape objects. Note, however, that the current scape objects could, nonetheless, be derived from either Mirror’s objects, current scape’s other objects, or another Scapes objects as described in the ontologies.

END

The Scape implementation is done using a similar technology stack as the Mirror Core, and a M-V-C architecture. Every Scape shares a set of services to access and export these ontological components. Following pictures show snippets of the HealthGame Scrape in operation.
Using same process as followed for HealthGame, we describe the ontology specifications and configuration for the other Scapes (sections 3.2-3.4), and implement them with the Mirror framework. In the following sections 5.2 and 5.3, we provide only the domain ontology and domain mappings for these Scapes. Other configuration and specifications are described in a similar manner, but are not specified here to avoid unnecessary digression, because they are not used to demonstrate the framework operational validation.

### 5.2 GreenCity Scape

Domain Ontology has objects and relations as shown in the figure below. The analyst is mapped to User in Mirror, while Energy, City and Equipments (like Transmission Boosters, Windmills and Energy Sinks) are modeled as Assets in the Core.
Figure 28: GreenCity Domain Ontology

Figure 29: GreenCity Scape Implementation
5.3 RealtyHub Scape

The domain ontology for RealtyHub implementation is shown below.

![RealtyHub Domain Ontology](image)

Figure 30: RealtyHub Domain Ontology

The domain mapping is specified as: Customer mapped to User in the Core, while Quotes and the Property are mapped to Assets in the Core.

![RealtyHub Scape Implementation](image)

Figure 31: RealtyHub Scape Implementation
5.4 Framework Operational Validations

With all the Scapes described and implemented in the Mirror framework, along with the Mirror Core, this section describes how the framework features are used to derive the missing interconnect identified in section 3.5. It also provides operational validations for the framework as specified in section 2.3.

Using the publishing and data sharing services of the Core, the Scapes push their operational data to the Mirror Core. The Core uses its mediation services to map that data into its vocabulary, and then uses the ontology specifications of the Core and Scapes to populate the instances of the concepts and rules defined in the Scapes. This is done with all the ontology specifications for all Scapes, and all this knowledge is processed together. The result of this process is depicted in figure 30 below. The red highlights are example outcomes of this process, and they associated with white textboxes to convey the information through the picture.

![Image of rule execution with shared upper ontology in the Core](image)

**Figure 32: Rule Execution with Shared Upper Ontology in the Core**

With operational data from Scapes and a vocabulary to understand it, the Core applies the rules defined in the Scape configuration files to the operational data and infers any
implications of the interactions across Scapes. This is the process that helps make the tacit knowledge in individual Scapes explicit throughout the Mirror, and leverage knowledge from other collaborations and domain software. This inference is done currently using an OWL reasoner. We used Hermit and FaCT++ with Protégé editor to demonstrate this inference below in figure 32; however, an inference engine with an OWL API is integrated in the Mirror Core as shown earlier in section 4.6.2.

![Figure 33: Understanding and Expressing Scape Objects & Behavior](image)

In the above figure, the red highlights represent the inferred knowledge. Here, for example, the Core had received operational data from HealthGame Scape that a user “user123” has alcohol level 1.8, and has chosen to drive home. The dynamics of a user being drunk and pulled over by a cop are defined using HealthGame domain ontology, and are available to the Core as shown in Figure 31. The Core uses this Scape specific knowledge to infer the state of user “user123” in HealthGame, which is “pulledOver”. It also used the domain mapping ontology to figure out that user123 has bought some gyms (assets) and has submitted a request called “finReq1”. This is how the Core “understands” the Scape dynamics.
The Core also inferred knowledge about the power station that powered the city, whose energy management was simulated in GreenCity. Because GreenCity simulated energy usage for Columbus that is beyond efficient limit of the serving power station (this is shared via the operational data of the Scapes), Core inferred that the power station that powers Columbus, located in Chicago, is now overloaded. This is shown in figure 33.

![Figure 33: Overloaded Power Station](image)

Naturally, since the power station is located in Chicago and pollutes more, Chicago is now being polluted more. This is a classic scenario of intrinsic domain interconnection in the real world, and the city being polluted is very crucial information that should be made available to the users.

After the rule execution and inference process, the Core makes all this otherwise tacit knowledge available to Scapes. The RealtyHub picks up this data and now alerts its users whenever any purchase transaction around Chicago is being finalized. This implementation is shown in figure 35 below. Similarly, HealthGame can pick this up to model new scenarios in Chicago.
Figure 35: Enabling the Domain Interconnection

Figure 36: Enabling the Domain Interconnection
Following are some scenarios that help demonstrate how the framework enables service discovery and component re-use. The “search” mentioned below is implemented by the API integrated in Mirror Core.

- How should I model a user in my Scape, closest to that in Scape X?
  
  ➔ *Export Schema, Export Data*

- What GIS Features can I re-use?
  
  ➔ *Search Service ontology for “GISEnabledBy”*

- What process can I re-use to index this Terabyte data?
  
  ➔ *Search Service Ontology for invoke-able processes*

- What parts of farming are currently already modeled? (What more do I need to do?)
  
  ➔ *Lookup Upper Ontology, Search Respective Scape Domain & Service Ontology*
6. Summary of Validations

6.1 Framework Design Validations

6.1.1 Completeness of the Shared Upper Ontology

The shared upper domain ontology is dynamically extensible, while the completeness and sufficiency of the upper dynamics ontology is validated in section 4.4.1.

6.1.2 Applicability of the Framework

Chapters 4 and 5 presented the Mirror Core implementation, and implementation of 3 different Scapes with the Mirror Core. Section 5.4 also demonstrated the features of Mirror.

6.1.3 Sharing & Re-Use

Chapter 4 and 5 demonstrated the sharing of knowledge, data and schema for rule execution and re-use. It also shows how the framework enables service discovery, composition & configuration, and re-use.
6.1.4 Evolution & Road Map

Chapter 4 detailed the qualities of the framework. The flexibility and extensibility are the key qualities that allow incorporating change in the framework.

6.1.5 Framework Qualities

Section 4.5.2 details and justifies the following qualities of the framework: Extensibility, Flexibility, Interoperability and Security.

6.2 Framework Operational Validations

6.2.1 Establishing a Shared Understanding of the Underlying Reality

Chapter 4 presented the shared upper domain and dynamics ontology, and demonstrated the sufficiency through section 4.4.1

6.2.2 Scape Configuration & Development Process

Chapter 5 implemented the process of Scape configuration and development, and successfully developed 3 Scapes with the Mirror framework

6.2.3 Understanding and Interconnecting Domain Objects & Behavior

Section 5.4 demonstrated how the framework enabled understanding domain objects, dynamics and how the inter-domain implications of intra-domain user actions can be made explicit.
6.2.4 Demonstrating Sharing & Re-Use

Chapters 4 and 5 present the services that enable sharing and reuse of components in the Mirror framework; section 5.4 demonstrates the sharing and reuse within the framework.
7. Conclusion

The Mirror framework helped to express and interpret the underlying, shared commonality between different domain software. We were also able to incorporate traceability, not just between configuration and programming work-products, but also between the domain goals and decisions that each Scape modeled. Owing to the design of the framework, it exhibits following properties that are primal to sustaining any cyberinfrastructure:

- Flexibility and Extensibility
- Interoperability
- Commonly understandable representation of heterogeneous knowledge
- Configurability & Declarative Programming (Reduced Scape Development workload)
- An Innovation Loop for Feedback and ITILv3 type Continuous Development that is enabled by Evolution

Overall, the framework served both – as software for the enterprise and as an enterprise for the software to evolve.

Note again, that the enterprise in this context is the mirrored world, as mentioned in section 1.3.
The framework helped to semantically interconnect the dynamics of any distributed simulation, training, gaming or other domain software with another, possibly across enterprises with differing business goals and domains, to get away with implicitly modeled assumptions and ignorance about factors seemingly external to the software. Consequently, we could dynamically combine the user decisions and derived knowledge from any and multiple domain software operating in different, distributed environments, for a decision-making or learning ability more consistent with real-world constraints.

This helped us take a first step towards realizing an end-to-end solution that spans different domains to better model day-to-day human (inter)actions.
8. Future Work

Future work should involve extension of the framework along following directions:

- Expand and formalize the syntax for the Scape API files, implement a Parser for the API file
- Incorporate a full power search ability on the configuration data objects and ontologies
- Incorporate rule interpretation and execution ability in the Framework, not only to further the understanding of Scapes but also to help evolution and integration [9, 17].
- Incorporate a full ability of logging and analytics with more hooks in the system for monitoring purposes
- Implement the semi-automation for Scape development process, and persistence for Ontologies as 4.7.
References


[18] Suzetter Stoutenburg, Leo Obrst, Deborah Nichols, Paul Franklin, Ken Samuel, Michael Prausa, “Ontologies in OWL for Rapid Enterprise Integration”


[31] Eric Wang, Sung Ah Kim, Yong Se Kim, “Case Study in Rule Inferencing using Protégé and Jess”


[33] Cheyney Loffing, “Exploring a Framework for Goal-Driven Collaboration through Serious Gaming”

[34] IPSV Board, “Integrated Public Service Vocabulary” (IPSV), version 2.00

[35] Yogish Pai, SOA Blueprint


[37] Brian Kahin, “Cyberinfrastructure and Innovation Policy”

[39] Christopher J. Mackie, “Cyberinfrastructure, Institutions and Sustainability”

[40] Peter A. Freeman, “Is ‘Designing’ Cyberinfrastructure – or, Even, Defining It – Possible?”

[41] Etienne Wenger, “Communities of Practice: Learning, Meaning and Identity”


[44] Hoh In, Rick Kazman, David Olson, “From Requirements Negotiation to Software Architectural Decisions”


[51] André van der Hoek, Institute for Software Research, UCI, “Capturing Product Line Architectures”


[58] Roger Eggen, Suresh Sunku, “Efficiency of SOAP versus JMS”


Appendix A: SRMN Solution Design & Implementation Details

The Governance Framework & Enterprise Organization

- Organizational Hierarchy

In the case of SRMN, it is evident that an open-flat collaboration model would lead to chaotic situations and will create performance and consistency bottlenecks. The organization must therefore provide a way to regulate the change to the ontology or meta-model in a hybrid open-hierarchical system. In an attempt to define precise roles and responsibilities and establish a clear workflow process over them, we separated the core change management activities from the tasks that require research efforts and inputs.

We established a sub-organization within the SRMN, consisting of CETI that will coordinate the collaborative effort while working on the Change Management process; and call this core organization the STEM Model Management Partnership (SMMP). Figure 2 depicts the enterprise that we are dealing with. The SMMP takes on the governance of the model and the processes around the model. It relies on the SMMP Steering group for outreach and supervising purposes. Within the SMMP, we form two working teams supervised and coordinated by the “Configuration Management Chair” who is the final authority. The two working teams are: the Configuration Control Board (CCB) and the Research Review Board (RRB).
**CCB Role:** The Configuration Control Board (CCB) is an organization of advisory board, system model engineers, and help desk team that manages and performs the change control activities. The CCB is inclusive of a variety of expertise dealing with routine technical bugs in the application, improvements in the existing model, incorporating new elements in the existing model, providing technical support for the system and its applications and so on. The CCB takes care of formalizing the change proposal / contribution to the model, defining user obligations, allocating and assigning resources to work on it, track the progress, report back to the contributor, receive feedbacks about the change process and its implementation, and most importantly, incorporate a continual improvement process depending on peer, contributor or other feedback. In summary, it is responsible for maintaining an up-to-date, stable model configuration. The CCB is well versed with the technical aspects of the model and is also responsible for recruitment of individuals in its own group.

---

*Source: Effort by STEM Model Management Partnership (SMMP) at Ohio State University, Columbus*
**RRB Role:** Approval decisions by advisory board that require validation of existing research and need extensive knowledge of the three aspects of education, technology and policy behind the model, are delegated to the Research Review Board (RRB) in their (CCB’s) workflow. The RRB is a team of members that are adept in research aspects of the model; they have better understanding of the bigger picture and limitations/enhancements/features of the model. This team is able to validate all aspects of a particular research over the model and decide if the new results are to be incorporated into the model, in conjunction with expertise from outside the SMMP if needed. As the CCB needs, it taps on the RRB expertise for inputs to proceed with or stop processing a change proposal. These interactions are depicted in Figure 3. The CCB and RRB are supervised by a Configuration Management Chair and the operation and strategy of this whole organization is steered by the political, technological and social influence of the SRMN steering group.

![SRMN Governance Workflow and Initial Solution](image)

**Workflow and Initial Solution**

The workflow process used is derived from the generic ITIL change management process definitions. Formalizing it, we require a contributor to populate a pre-formatted semi-structured change proposal template for proposing a potential change or enhancement to the
existing model. The template is currently pronounced to include following details about the change proposal:

- **Goal/Purpose**: Textual description by contributor.

- **Type**: Type of change – the contributor’s perception of the change among select types.

- **Target**: The logical sub-part of the model targeted, or the conceptual aim targeted.

- **Impact**: The possible impact to the other parts of the model, the overall result achieved.

- **Description**: Any other details the contributor might wish to add.

This template is then reviewed by the CCB and is categorized into following request types:

- **Bug Report (BR)**:
  
  - Requires documentation of conditions, symptoms, errors, recommended fix (optional) in the Description

- **Technical Upgrade (TU)**:

  - Requires detailed technical rationale, impact (benefits)

- **Research-Driven Change (RD)**:

  - **RD-A**: Contributor provides data + new model elements. Requires:
    
    - Detailed rationale/case statement, Complete data sources, Required new model elements, Linkages & dependencies to existing model identified, Proof of functional verification

  - **RD-B**: Contributor provides data + relevant description, doesn’t have expertise to design new model elements. Requires:
- Complete data sources, Justification of research, a case statement, Detailed description of the required change

The configuration management process is currently designed to follow the phases shown in Figure 4. (Legends: ◊ denote Associated Roles in the description later)

The workflow is described in detail below.

1. Initial Screening ◊ CCB
   - Verify that the template requirements are met
   - Review the change proposal
   - Categorize the change (based on scope {BR, TU, RD-A/B}, complexity, impact)
   - Record the change status

2. Assessment ◊ CCB+RRB+CM Chair
   - Assign/coordinate BR, TU technical review/ recommendations
   - Assign/coordinate RD-A/B research review via RRB
   - Assign/coordinate technical review of validated RD
   - Consolidate assessment report; record the status

3. Disposition ◊ CCB
   - Receive reviewed changes
   - Evaluate against criticality, impact, etc.
   - Approve/reject change proposals
   - Report and record detailed change status
4. Plan \(\Diamond\) CCB
   - Appraise implementation scope, complexity
   - Evaluate against calendar & resource constraints
   - Evaluate against current version/upgrade scheme
   - Map approved changes into workflow
   - Assign/coordinate implementation resources
   - Report the plan documents, record detailed change status

5. Implement \(\Diamond\) CCB
   - Incorporate changes into the sub-model: test branch
   - Verify the change
   - Simulate integrated model, verify results
   - Merge the change into full model
   - Record change status, Report verification results

6. Close \(\Diamond\) CCB
   - Issue new patch/release
   - Consolidate evaluation report, report appropriate results to users
   - Verify configuration items are up-to-date

Figure 39: SRMN Roles and Workflow
- Verify traceability: goal-implementation
- User feedback survey

Figure 40: SRMN Solution Implementation
Appendix B: Mirror Core Implementation Details

Sample Subset of Use Cases for Mirror (Initial Prototype):

These use cases set a requirement to simulate the basic world interactions and are a subset of the use cases used for the Mirror Framework development.

**Actors:**

1. Anonymous User
2. Registered User
3. Group Member (a Registered User)
4. Group Admin
5. Group Owner
6. System Admin

**Sample User Stories as a part of the Agile Development Iterations:**

As an Anonymous User or Registered user:

1. A user views the Scapes and its details on.
2. A user searches a Scape on campus to attend it.
   
   a. User uses free text/interactive forms/voice for entering or refining his preferences.

3. Route selection
   
   a. A user selects the source and destination to search the optimal route at that time.
   
   b. The system responds with the options for selecting routes like most scenic route, least traffic etc.
   
   c. A user selects one or more criteria for finding route.
   
   d. System responds with the most optimal route based on the selected criteria

4. A user rates a route or Scape (based on criteria like traffic, time, road conditions, current Scapes etc) to facilitate better transportation for himself and others. (based on Scape)
   
   a. A user selects a route to rate
   
   b. A user defines the criteria for rating (predefined factors like scenic route etc)
   
   c. A user enters the route rating
   
   d. A user adds a comment on the route

5. A user views other users’ ratings for decision making or getting information.
   
   a. A user selects a particular route
   
   b. The system displays the route using location services
   
   c. The system displays the overall route rating
   
   d. The system displays the ratings by users – aggregated.
e. The system displays the comments by users.

6. Can read/search the announcements based on keywords, tags, etc.

7. Anonymous user signs up to create an account in the system filling his information

As a Registered User

1. Logs in to the system to use it
   a. User enters id and password to login to the system.

2. The user enhances/modifies his presence through managing his profile
   a. The user creates/updates a basic profile (Name, contact, uname, psswd, location)
   b. The user creates/updates advanced profile based on his preferences or based on Scape (optional)
   c. System mines profile from user.

3. Perform transactions with the system/another user:
   a. Submits information to the system to amend or enhance its existing asset data
      i. Upload files
      ii. Enhance the map information (suggest add landmarks, locations on map)
      iii. Tag images to map locations
      iv. Write a comment suggesting some improvement in system
      v. To use (like going through a building or through a park etc, based on the Scape)
b. Donate an asset to another user

c. Exchange an asset with another user for a value defined by the asset owner

4. Manages a Scape and its details to publicize it on the “public notice board” and everybody’s homepage (User checks “publicize” check box for e.g. to add it to the public notice board).

   a. Enters the name/title of the Scape

   b. Enters Scape location

   c. Enters Scape description

   d. Enters Scape date and time

   e. Enter any other comments for the Scape

   f. Scape type

      a. Competition

      b. General Convening

   g. Scape Rules

      a. Select constraint-value pairs: (time, speed, place, area, distance, cardinality, etc)

      b. Add a constraint and its value: upload size limit, no. of participants per group, no. of groups, transportation mode, etc.

      c. If type == competition

         i. Winning criteria (constraint-value)

      d. Disqualification criteria (constraint-value)
OR Textual description for above all

e. Participation criteria (registered users, anonymous users, private: by invitation of participants)

h. Creates sub-Scapes within Scapes

i. A user deletes a Scape to cancel it

5. Registers for a Scape (Subscribe to this Scape check box for e.g.)

6. De-registers from a Scape to discontinue participation.

7. Selects the viewing options (themes/displays) for a Scape (possibly a Future implementation item)

8. Manage announcement

    a. User adds an announcement (anything) to publicize it with its details (pure text with char limit, links)

    b. Edits the announcement details to update them.

    c. Inactivates the announcement

9. Votes the Scape ‘thumbs up’ or ‘thumbs down’.

10. Adds a comment on the Scape.

11. Views the feedback received on a Scape based on information aggregation like weather conditions, traffic, other Scapes, response from other users, etc (future/past).

12. Votes the comment ‘thumbs up’ or ‘thumbs down’.

13. Searches groups that are listed public by their respective owners.

14. A user forms a group by filling out the group details to generate the group profile:
a. Name
b. Cardinality limit
c. Accessibility (anonymous, registered users, private)
d. Owner(s) (default: requestor user)
e. Governing mechanism for group

15. Manage self membership
   a. Joins a group to participate in it.
   b. Views his memberships.
   c. Leaves a group to discontinue participation.

16. Sends and receives messages (in user's scrapbook) to communicate with another user.
   a. A user writes a message for another user to view.
   b. Deletes any of his messages.
   c. A user views the messages sent to him by other users.

17. A user communicates on team forums to collaborate with a set of users (depending on the accessibility of the group as set by the owners)
   a. A user writes a message on forum for everyone in the group to view.
   b. A user views the message posted by other users.
   c. Deletes his message on the forum.

As a Group Member:

1. Invites a user to a restricted access group.
2. A user communicates on team forums to collaborate with a set of users (depending on the accessibility of the group as set by the owners)
   
   a. A user writes a message on forum for everyone in the group to view.
   
   b. A user views the message posted by other users.
   
   c. Deletes his message on the forum.

As a Group

1. Manages a group owned Scape and its details to publicize it on the “public notice board” and everybody’s homepage (User checks “publicize” check box for e.g. to add it to the public notice board).
   
   a. Enters the name/title of the Scape
   
   b. Enters Scape location
   
   c. Enters Scape description
   
   d. Enters Scape date and time
   
   e. Enter any other comments for the Scape
   
   f. Scape type
      
      a. Competition
   
      b. General Convening
   
   g. Scape Rules
      
      a. Select constraint-value pairs: (time, speed, place, area, distance, cardinality, etc)
   
      b. Add a constraint and its value: upload size limit, no. of participants per group, no. of groups, transportation mode, etc.
c. Participation criteria (registered users, anonymous users, private: by invitation of participants)

h. Creates sub-Scapes within Scapes

i. A user deletes a Scape to cancel it

2. A user manages the group by editing the group profile
   a. Name
   b. Cardinality limit
   c. Accessibility (anonymous, registered users, private)
   d. Governing mechanism for group

3. Adds a member to the group

4. Kicks off a user from a group

5. Creates/Disables forums and defines accessibility to it to help others access/not access the forum

6. Censors member activities

7. Manages (sells/buys/exchanges/donates) the group of assets.

8. Deletes the group


10. Removes an admin.

11. Changes the governance structure of the group.

As an Evaluator, a Registered User:
1. Evaluates the submissions and results of users participating in a competition type Scape.

2. Reports his/her judgment on the announcement board

As a system Admin,

1. Sends membership deletion warnings to inactive members

2. Purges inactive users out of the system with prior warnings (weekly warnings)

3. Generates random system events in the system.

4. Notifies appropriate events to members/Scapes as warnings on announcement board.

5. Responds to a Scape or event (System or User generated).

6. Logs Scapes/events affecting mirror world for future references.

7. Posts appropriate responses on announcement board.

8. Defines basic Laws for Behavior, transaction and competition.

9. Penalizes and Rewards user contribution

10. Stores the user information, event/Scape information, etc; captures the world-Scape details for users to see – populates user profile (system-events, customized home page elements, preferred announcements, etc)

11. Remembers the preferences of each user for display, interests, etc

12. Maintains a log of user activity to extract trust ratings.

13. Maintains ratings on comments, events etc as per the user ratings.
### Governance Structure

<table>
<thead>
<tr>
<th>Group</th>
<th>Governance</th>
<th>Public</th>
<th>Democracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Deletion</td>
<td>Owner</td>
<td>Owner/Admin</td>
<td>Quorum of Administrators / Owner / Administrator</td>
</tr>
<tr>
<td>Governance Change</td>
<td>Owner</td>
<td>Owner/Admin</td>
<td>Quorum of Administrators / Owner</td>
</tr>
<tr>
<td>Add Members</td>
<td>Owner</td>
<td>Administrator</td>
<td>Administrator / Owner</td>
</tr>
<tr>
<td>Delete Members</td>
<td>Owner</td>
<td>Administrator</td>
<td>Administrator / Owner</td>
</tr>
<tr>
<td>Manage Group-owned Scapes</td>
<td>Owner</td>
<td>Administrator</td>
<td>Quorum of Admins / Owner</td>
</tr>
<tr>
<td>Add Admin</td>
<td>-</td>
<td>Owner/Administrat or</td>
<td>Owner/Admin/Quorum of Admins</td>
</tr>
<tr>
<td>Remove Admin</td>
<td>-</td>
<td>Owner/Administrat or</td>
<td>Owner/Quorum of Admins</td>
</tr>
<tr>
<td>Censor Member Activity</td>
<td>Owner</td>
<td>Administrator</td>
<td>Owner/Administrat or</td>
</tr>
</tbody>
</table>

**Table 3: Mirror Prototype Governance Structure**
User Stories Prioritization and Iteration Management for Mirror Implementation:

Figure 41: Core User Stories - Prioritization and Iteration Management

Sample Use Case Diagrams:
Figure 42: Mirror Core Sample Use-Cases-1

Figure 43: Mirror Core Sample Use-Cases-2
A snapshot of the Database Schema (used to derive and relate to the object interaction model)
Figure 45: Shared Ontology used to derive the Core Schema

Arrows represent relationships between the objects
### Schema Description:

<table>
<thead>
<tr>
<th>TABLE NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Scape</td>
<td>All the Scape details, references to presence, owner/creator User, and has multiple Scape Rules</td>
</tr>
<tr>
<td>2. Presence</td>
<td>A geographical location at a time, and orientation data holder</td>
</tr>
<tr>
<td>3. Constraint_table</td>
<td>Specifies the constraints that a rule can apply. This will be derived from the Mirror core rules and the Scape configuration file rules.</td>
</tr>
<tr>
<td>4. Event</td>
<td>Holds the information about ‘event’</td>
</tr>
<tr>
<td>5. Event Rule</td>
<td>Specifies what rules to trigger in case an event occurs, and what consistency constraints should be applicable in the context of that event.</td>
</tr>
<tr>
<td>6. Scape Rule</td>
<td>Specifies what consistency constraints are applicable in the context of a Scape</td>
</tr>
<tr>
<td>7. Asset</td>
<td>Any object (not user) in the mirror, has geoinformation, and has relation to other assets</td>
</tr>
</tbody>
</table>

Continued

*Table 4: Core Schema Description*
Table 4 continued

<table>
<thead>
<tr>
<th>No.</th>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.</td>
<td>Value</td>
<td>Stores the value of an asset owned by some user or group, in the context of a Scape. (Note: the way a Scape interprets the value may be different; but the actual ‘value’ of an asset is the property of mirror than any one Scape, so it is consistent.)</td>
</tr>
<tr>
<td>9.</td>
<td>State</td>
<td>Maintains the state of an asset in different Scapes. E.g. a Stadium might be used in one Scape as a shelter. It might be ‘half-filled’, or a house may be ‘occupied’, a theater may be ‘full’, etc.</td>
</tr>
<tr>
<td>10.</td>
<td>GeoRelation</td>
<td>Inter-relation (orientation and geography-wise) between assets</td>
</tr>
<tr>
<td>11.</td>
<td>OwnedBy</td>
<td>Stores what Asset is owned by which user/group</td>
</tr>
<tr>
<td>12.</td>
<td>User</td>
<td>Stores basic user information</td>
</tr>
<tr>
<td>13.</td>
<td>Group_table</td>
<td>Group information – owner/creator/administrator</td>
</tr>
<tr>
<td>14.</td>
<td>Membership</td>
<td>Stores the user-group memberships</td>
</tr>
<tr>
<td>15.</td>
<td>UserProfile</td>
<td>Stores all the profile information of user</td>
</tr>
<tr>
<td>16.</td>
<td>UserRelation, friend</td>
<td>Relation between users (friend, spouse, neighbor, enemy, etc)</td>
</tr>
<tr>
<td>17.</td>
<td>Role</td>
<td>Description of a responsibility that a user carries out, in context of some Scape interaction</td>
</tr>
<tr>
<td>18.</td>
<td>Interaction</td>
<td>Any correspondence between two users or user and the system.</td>
</tr>
</tbody>
</table>
Table 4 Continued

| 19. | InteractionContains | The assets that might be involved in an interaction – the agents or entities as we model in runtime interaction diagram |
| 20. | InteractionType | Type of interaction: e.g. Communication, upload, enhancement request, feedback, etc |
| 21. | InteractionInvolves | Stores what users taken part as what roles in which interaction, and under what Scape context |
| 22. | InteractionType | Stores what type of interaction occured |
| 23. | UniversityProfile | Stores the university profile of a user, if he is affiliated to OSU. This is done for immediate usage in the games developed in OSU. |
| 24. | AssetGroupRelationship | stores what assets are owned by what groups |
| 25. | AssetUserRelationship | stores what assets are owned by what users |
| 26. | EventContains | stores what interactions occurred in what events |
| 27. | EventRating | Stores a basic feedback structure for events |
| 28. | ScapeGroupRelationship | Stores the group –Scape ownerships |
| 29. | ScapeRating | Stores a basic feedback structure for Scapes |
| 30. | ScapeUserRelationship | Stores Scape – user ownerships (what Scape is created by which user) |

**Mirror Core Runtime Snapshots:**
172

Adobe Flex Components:

Figure 46: Mirror Core Operation

Figure 47: Core Flex (View + Controller) Code Structure
Figure 48: Core EJB (Model) Code Structure
Appendix C: A study of Cyberinfrastructures

Cyberinfrastructures (abbr. CI)

Contemporary CI, without loss of generality, have fairly common goal of enabling research and development through collaboration and high outreach. Following are some of the features of the environment in which a typical CI operates:

- Multiple Stakeholders

Owing to the wide scope of areas – technologically and socially - that it touches, a cyberinfrastructure naturally attracts interests of and forms a part of various initiatives at once. This pulls in different stakeholders with different interests, a result of which has been incorporation of performance management and metric monitoring at different levels of measurements (physical, software, system, business) and instrumentation of data flowing through the system to generate reports for feedback and improvements. For e.g. the CI that CETI at Ohio State University is has built for SRMN has 3 major stakeholders: industrialists, educationalists and researchers.

This study is derived collectively from case studies mentioned as cyberinfrastructure examples below in table 5, and the cyberinfrastructure background research identified in section 1.3.
• Multiple Communities & Complex Interactions
Any architecture for shared development invites contributors from across places and these contributors need to collaborate electronically in order to produce quality outcomes. With interaction systems like mail, instant messaging, discussion forums, etc all of the interactions are rarely captured and tacit knowledge is hardly represented in the system. E.g. Google groups, Yahoo groups, Google Wave

• Multiple data-sets with historical lineages
Cyberinfrastructures often serve needs in-the-large and that requires managing massive datasets, from heterogeneous sources that must be linked for experimentation, simulation and observation. E.g. Environmental Cyberinfrastructures

• Visualization, Virtual environment & Simulation Power
Typical simulations require imitation of the real world environment to a considerably large extent. The notion of real “world” in this sense is not quite restricted to world on earth, but extends from its core to crust, and from inner space to outer space as well. Common examples include Environmental and space cyberinfrastructures (Neptune, ALMA, Sloan Digital Sky Survey, etc) games like second life, and biological systems like “Genomes to LIFE”.

• Information Assurance
Ensuring the confidentiality, integrity, authentication, availability and non-repudiation of information, as is for cyberinfrastructures, is a routine consideration for all IT systems.

A generic cyberinfrastructure can therefore be seen as a “complete” (ideal) system combining sciences of software development, human collaboration, tool integration, data and knowledge management, computation power and performance management into one single environment with high accessibility and flexibility. Cyberinfrastructures, however, cannot be “defined” [40] using such verbiage because they are rather human-centered and application specific. Listed below are some of the cyberinfrastructures serving different purposes, with some of their predominant features.
Exemplary Cyberinfrastructures:

<table>
<thead>
<tr>
<th>Purpose &amp; Examples</th>
<th>Typical Governance model</th>
<th>Individual Use Cases</th>
<th>Integrated Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function primarily as collaboratories for enhancing or improving an abstract ontology or meta-definition of an object or object itself through community contributions. Some extend this idea to handle multiple such projects at a time and providing the developers/researchers a place to work together to achieve the end result. E.g. Collaborative development sites like joomlacode.org, stemnetwork.org</td>
<td>Standards/ Open source operation, typically hybrid model of governance</td>
<td>Role-based and Community-based access control, Change Management, Traceability,</td>
<td>Process Frameworks, Project Management tools</td>
</tr>
</tbody>
</table>

Table 5: Example (existing) Cyberinfrastructures
Focus on knowledge accumulation and triage through the enterprise, regulate change management, incident management, and supports integrative viewing of tools used in the enterprise.

E.g. Sophisticated Customer Service Systems, like 311 in the City of Columbus, or 911 system in USA.

<table>
<thead>
<tr>
<th>Focus on knowledge accumulation and triage through the enterprise, regulate change management, incident management, and supports integrative viewing of tools used in the enterprise.</th>
<th>Typically Centralized: Need enterprise-wide management mechanisms, IT-Business tracking capability and traceability management.</th>
<th>Data acquisition, change/incident management, tool integration, data visualization Role-based access</th>
<th>Tracking &amp; Traceability Tools, Data collection</th>
</tr>
</thead>
</table>

Continued
| Provide infrastructure for high-end computations to a varied audience, and have grid-computing and high performance computation power as their core operational features. E.g. New York State Grid (NYS-Grid) ([http://nysgrid.org](http://nysgrid.org)) by the NSF through grants to colleges. | Typically Centralized (or Hybrid): Because such CI(s) focus more on VOI i.e. Value on Investment (through supporting innovation and knowledge generation) than innovation in the CI itself. | High Concurrency & Consistency, Data and Code Parallelization | Parallelization tools for distribution and executing serial codes in parallel manner |
| Make pieces of information ubiquitous and easily accessible to all users; such systems concentrate on data acquisition, data management and visualization. E.g. NSF’s initiatives through its Advisory Committee on Environmental Research & Education (AC-ERE), called as Environmental Cyberinfrastructure (ECI) ([http://www.nsf.gov/geo/ere/ecweb](http://www.nsf.gov/geo/ere/ecweb)). | Typically Hybrid (or Decentralized) governance. | Data consistency, Acquisition & Visualization | Sensory channels, Data Management tools |
**CI Business Case**

Summarizing from the above examples, following are a few “reasons” that help justify the need to build a cyberinfrastructure:

- Increase the VOI through effective and shared use of appropriate technological, knowledge and infrastructural assets
- Increase the advertising ability by increasing outreach
- Embrace ability to sustain, and respond to, disruptive innovation in the market
- An end-to-end solution, keeping generic the methods for inputs, outputs and processing, to solve complex problems related to different fields

The ideal and most functional test of a cyberinfrastructure would be dynamic service composition in a non-routine request environment, where the cyberinfrastructure has to cater to differing needs for different sets of users, providing different view and knowledge of a fairly common underlying data set.

**CI Evolution, Research issues & Architectural Challenges**

CI(s) that aim at enabling and supporting research initiatives should be designed to embrace the change identified and the results achieved from the current efforts. Hence, understanding the performance, features and collaboration as the infrastructure is currently enabling, and the gap between what will emerge as requirement in the near and far future is necessary. However, this is specific to applications of cyberinfrastructure and is discussed in forthcoming text.

Emerging CI(s) cannot be completely characterized ahead of time, and there has to be a process that will enable the leadership to recognize and respond to the changing environment outside, a good example of the situation driving this is given in [36]. However, a thorough study of characteristics required and goals envisioned is needed in the minds to set out correct direction while the initial efforts. The process should then embrace updates to technologies and strategies and apply to the current infrastructure and in-turn turbo-
charge the cyber-infrastructure to maturation. This boils down to developing precise but
generic enough use-case scenarios and requirements that:

- represent the spectrum of current engineering efforts (and not just specific project
  spaces)
- aid in setting priorities for evolution of the cyberinfrastructure
- suggest metrics and in-turn generate use-cases to instrument the metrics for
  monitoring the goal achievements
- develop, unfold themselves as the CI portfolio and the outside landscape changes
  (allow the CI to embrace flexibility)

Such a set of requirements should be used to drive the design, development and deployment
of the cyberinfrastructures that are aimed at enabling research and high collaboration.

Identifying initial requirements is, however, a rather static part in the overall evolution of a
cyberinfrastructure. The important part while building it is to identify research challenges
around the processes and targeting research efforts towards them to open up opportunities
to build next-generation abilities that the infrastructure can provide.

The fact that a cyberinfrastructure is an integrative solution renders the cyberinfrastructure
open to all challenges that its individual components pose, as well as the challenges of
integration and maintenance of the cyberinfrastructure as a whole [26]. Cyberinfrastructures
should be designed to enable following features in order to provide any robust solution:

- **Operational Management of Cyberinfrastructures**
  Cyberinfrastructures provide the technological and infrastructural platform for
  addressing the next generation science, engineering and research problems. They can
  span from stand-alone, minimal local resource hungry application to client based,
  distributed, resource heavy system in many dimensions. Being an integrative system,
  cyberinfrastructure operation has to address concerns including:

  - **Interoperability** – of resources, and software through and across
    cyberinfrastructure layers
  - **Evolution of middleware** as the CI requirements change
• Updates to the software and hardware components to stay current with technology
• Other maintenance operations specific to software application/hardware.

• Collaboration & Evolution Process Management for CI [6]
  • Coordination of different activities that the CI supports: e.g. development, deployment, research, etc
  • Co-opetition: Cyberinfrastructure enables and requires collaboration; but also provides competitive environments to the communities. There must be a controlled harmony between the competition and co-operation which is termed as Co-operative competition or “co-opetition”. The processes designed should make it evident and consequential that all efforts are ultimately oriented towards a common goal of advancing the science and engineering in the society.

• Maintaining equilibrium with ecology
  Change in business requirements is now a universal phenomenon, and with such changes the products are quickly rendered useless or less competitive if they do not adapt to the surrounding environment. A cyberinfrastructure is no exception to this. Keeping complex systems evolving and relayed to the environmental changes involves responding timely to changes in technology, changes in organization and changes in business strategies. Not only do technology changes bring in compatibility concerns, they are most likely to affect the throughput of any cyberinfrastructure because underlying physical infrastructure and the Middleware is its driving engine.

  An attempt at conceptualizing an architecture that is forgiving to limited variance in technology (by de-coupling the requirements specification from implementation) is shown in Figure 29.

  The idea is to separate the requirements and implementation of the significant CI Layers through the means of a specification language. This specification will include what exactly the layer needs in order to function: what inputs it consumes and what outputs produces, its dependencies, etc. This specification also serves as an API for extending the layer. With this specification as a guideline, technology can be used as an “enabler” to implement this specification.
Appropriate separation of layers is a very key dependency for this framework to be validated, identifying the correct set of (tightly-coupled) technologies to function together in a layer and others (loosely tied technologies) to sandwich the interface specification will increase the technology variance that the CI can embrace.

Organizational changes might or might not result in any modifications to the cyberinfrastructure unless the cyberinfrastructure includes (governance) processes defined on roles that match the organizational roles and workflow, rendering the coupling between the enterprise organization and cyberinfrastructure as the governing factor. Business strategies are essentially all that need the software and systems in an enterprise. As is business-IT alignment necessary while building IT systems, it is even more necessary in cyberinfrastructures to keep it aligned through the life of business for survival and competitive edge.
• Broadening participation – community building and evolution
With increasing span and pervasion of internet, science no more manifests itself and cannot be an individual or dominantly individual contribution. It has unfolded itself into such vast and high number of areas that it is practically unconceivable to maintain expertise in all fields with time, for any individual or a small group. Cyberinfrastructure attempts at problems from many such streams of science, and in itself is a harmonious combination of technologies derived from different streams of science. A CI inherently needs support and expertise of different communities working in different areas and having common interests. A typical CI, therefore, must establish mechanisms for various communities to participate and communicate with each other effectively. Different forms of collaboration (open/closed or flat/hierarchical), governance models (centralized, distributed, hybrid), knowledge sharing and access must be carefully considered and evaluated to implement the correct strategy for communication and collaboration for a specific Cyberinfrastructure.

• Simulation and Visualization
Getting simulations close to real world is not an easy task because, in itself, the scope automatically spans all the actions possible in real world. However, even after having identified the correct subset of actions to be imitated, the assimilation of data from the system and heterogeneous external sources, validation of the data, establishment of mathematical models for simulation and actual visualization of the data doesn't appear to be a problem with a COTS solution in place.

Another part of Visualization is systemic more than customer facing. The system and business metrics captured from the system operation need to be projected in various abstractions to extract knowledge about the system health, and business of the enterprise.

• Ontology management and semantics-enabled change management
Many cyberinfrastructures help manage collaborative development of some ontology and/or objects. The collaboration of contributors naturally results in multiple outcomes that are, in high probability, intersecting with each other. Some contributions align well together, and some don’t when these changes to the
ontology should be well regulated to avoid chaotic conditions arising due to inconsistency and lack of integrity. Having a generic framework for automating this regulation process will need intelligence and knowledge derived using ontology.

- **Knowledge Mining**
  “Knowledge”, in this context, can refer to the knowledge about system/application or the business that is enabled by the system in operation. The foremost challenge in Knowledge Mining lies in knowing what knowledge to mine; and often this roots down to plugging in monitors and loggers in the system to capture data. Different views of the data captured by these monitors serve the purpose of unveiling “system” and “business” knowledge. If the knowledge that needs to be mined by the stakeholders, customers, etc and the data that this knowledge abstracts from can be identified while the CI design, the data assimilation process will be much easier and extensible in the future.

- **Traceability management in a CI**
  Traceability management has turned into a classic problem in software industry as the software systems get more complex. A CI can be visualized as a collection of different software based on different technologies. However, all these software satisfy different interests of stakeholders in an integrative manner. Thus, the requirements now has much more dimensions than in usual SDLC – the stakeholder dimension (what we call the system dimension), the vertical dimension (what is derived by independent software into their own set of requirements), and the physical dimension (the requirements set on the infrastructure and firmware).

  Traceability therefore has to be considered and, most importantly, correctly translated in much more dimensions in a CI than in usual software development. Framework for such traceability management could be extended from ideas in existing frameworks for the standard SDLC process, but are not seen consolidated in the research market.

- **Sustaining the CI using an open source model – software, funding**
  - Including more able staff for efficiency
• Bringing right people at the right places

• Embracing open standards

• Merging processes and project initiatives by identifying commonalities

• Identifying industry problems solvable by the infrastructure and reaching out to appropriate industries for funding, post the NSF funding phase.

• Gratifying the Customer & Stakeholder over time
  The ‘acceptance criteria’ is a measurable achievement that has to be defined and fleshed out for each level of operation in the workflow in order to determine whether the criteria have been completely satisfied or not. To promote and nurture innovation and research through the cyberinfrastructure, it is critical to understand the attributes of CI, and what it (CI) will take to deliver what is wanted out of it (CI): CI is ultimately an infrastructure and should be treated as one. It is, at some point of time in operation, going to have outdated components and will need to be maintained effectively without disturbing the services offered. The criteria for success varies at different level in the organizational hierarchy, therefore correct traceability has to be maintained while implementing the CI and using the CI. Varied base of users need varied services, and bring in varied results. CI is an environment wherein one set of services will not fit all. There has to be constant evolution and/or improvement of services offered.

• Fueling the effort
  The development, deployment and maintenance of the cyberinfrastructure are practically solely dependent on the funds available to invest on them. It is therefore critical to carefully plan the existing budget and also plan for acquiring future funding avenues. In the context of SRMN, for example, following is the direction of this effort maintained to keep the activities going:

  • Build a system portfolio: the current available and built infrastructure, current results and research, how the research models the current education and flaws in it and how do we plan to build a system that will identify and attempt to fix the flaws.
• Address to potentially interested audience (Science & Education agencies) across the nation

• Develop Partnership Opportunities through various programs.

We used this study to understand the environment of operation for Mirror, and drive its design and architecture.