Organizational Modeling of Large-Scale Multi-Agent Systems with Application to Computer Games

Okreša Đurić, Bogdan

University of Zagreb

Faculty of Organization and Informatics

Bogdan Okreša Đurić

ORGANIZATIONAL MODELING OF LARGE-SCALE MULTI-AGENT SYSTEMS WITH APPLICATION TO COMPUTER GAMES

DOCTORAL THESIS

Varaždin, 2018

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Bogdan Okreša Đurić

ORGANIZACIJSKO MODELIRANJE VIŠEAGENTNIH SUSTAVA VELIKIH RAZMJERA S PRIMJENOM NA RAČUNALNE IGRE

DOKTORSKI RAD

Varaždin, 2018.

University of Zagreb

Faculty of Organization and Informatics

Bogdan Okreša Đurić

ORGANIZATIONAL MODELING OF LARGE-SCALE MULTI-AGENT SYSTEMS WITH APPLICATION TO COMPUTER GAMES

DOCTORAL THESIS

Supervisor:
Assoc. Prof. Markus Schatten

Varaždin, 2018

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Bogdan Okreša Đurić

ORGANIZACIJSKO MODELIRANJE VIŠEAGENTNIH SUSTAVA VELIKIH RAZMJERA S PRIMJENOM NA RAČUNALNE IGRE

DOKTORSKI RAD

Mentor:
izv.prof.dr.sc. Markus Schatten

Varaždin, 2018.

DOCTORAL THESIS INFORMATION

I. AUTHOR

Name and surname	Bogdan Okreša Đurić
Date and place of birth	2 February 1989, Smederevo, Serbia
Faculty name and graduation date for level VII/I	Faculty of Organization and Informatics, 9 September 2010
Faculty name and graduation date for level VII/II	Faculty of Organization and Informatics, 21 June 2013
Current employment	Faculty of Organization and Informatics

II. DOCTORAL THESIS

Title	Organizational Modeling of Large-Scale Multi-Agent Systems with Application to Computer Games
Number of pages, figures, tables, appendixes, bibliographic information	236 pages, 38 figures, 60 tables, 3 appendices, 157 items of bibliographic information
Scientific area and field in which the title has been awarded	Scientific area Social Sciences, scientific field Information and Communication Sciences
Supervisors	Assoc. Prof. Markus Schatten, PhD
Faculty where the thesis was defended	Faculty of Organization and Informatics
Thesis mark and ordinal number	148

III. GRADE AND DEFENCE

Date of doctoral thesis topic acceptance	20 July 2016
Date of doctoral thesis submission	10 September 2018
Date of doctoral thesis positive grade	27 November 2018
Grading committee members	Prof. Mirko Maleković, PhD Prof. Sandra Lovrenčić, PhD Prof. Slobodan Ribarić, PhD
Date of doctoral thesis defence	14 December 2018
Defence committee members	Prof. Mirko Maleković, PhD Prof. Sandra Lovrenčić, PhD Prof. Slobodan Ribarić, PhD
Date of promotion	4 May 2019

Markus Schatten was born in Vienna, Austria, on 27 September 1981. He graduated study programme orientation Information Systems at the Faculty of Organization and Informatics in 2005. He obtained his master's degree in 2008 at the same faculty with the thesis entitled "Zasnivanje otvorene ontologije odabranih segmenata biometrijske znanosti" under the supervision of Prof. Miroslav Bača, PhD, and Prof. Mirko Čubrilo, PhD. He defended his doctoral thesis in 2010. on the topic "Programming Languages for Autopoiesis Facilitating Semantic Wiki Systems" under the mentorship of Prof. Mirko Čubrilo, PhD, and Prof. Miroslav Bača, PhD.

He started working as a Teaching Assistant at the Faculty of Organization and Informatics in 2006, as a member of the Department of Theoretical and Applied Foundations of Information Sciences. In 2010 he was promoted to Senior Teaching Assistant, with another promotion, to Assistant Professor, in 2011, all the while being a member of the same department. Along with being promoted to Assistant Professor, he was promoted to the title of a Scientific Advisor, scientific area of social sciences, scientific field of information and communication sciences, in 2013.

He was a part of the teaching staff in a number of courses on doctoral, master, and bachelor levels at the Faculty of Organization and Informatics, and held some classes at the Faculty of Information Studies in Novo Mesto, Slovenia, University of the People, USA.

He is the head and the founder of the Artificial Intelligence Laboratory at the Faculty of Organization and Informatics, and, since 2014, a member of the management board of the Intelligent Transport Systems association.

He authored or co-authored more than 80 scientific and professional publications, mentored more than 50 bachelor and master thesis and mentored or co-mentored five doktoral theses.

A big thank you to my parents and my sister!

For continued support and PhD-related chat and advice,
thank you to Markus, Igor, Petra, Martina, and Tonimir.

For always being there for me,
and enduring my lamentations and other (emotional) outbursts,
thank you to Ozano, Zrinka, Vedran, Goran, Andrija, and Kristijan.

A special thank you to all my international colleagues I was in contact with during the last couple of years, who made the whole experience that much better and richer!

And a big shout-out to all the others who had an impact on my journey thus far.

"All we have to decide is what to do with the time that is given us."
— J.R.R. Tolkien

This work has been fully supported by
Croatian Science Foundation
under the project number:
HRZZ-UIP-2013-11-8537.

Abstract

Abstract in English

The most popular and frequent methods of conducting a system of agents, of small- or large-scale, are those based on swarm intelligence, and organisational models. Organisational models for multi-agent systems are being developed alongside their role in the modern world. Technological improvements lead to creation of systems comprising thousands, or millions, of agents – large-scale multiagent system (LSMAS). Numerous LSMAS application domains (Internet of Everything (IoE), massively multi-player online games (MMOGs), smart cities, etc.) make LSMAS a genuinely useful concept in the modern era. Recent studies argue higher efficiency of LSMAS with imposed organisation, as opposed to systems with emerging intelligence. This makes organisational modelling of LSMAS a particularly interesting research subject. Organisational model based on ontology comprising LSMAS-related organisational concepts, built conforming to modern organisational perspectives for LSMAS, is a step towards easier LSMAS modelling. The ontology is basis for an organisational metamodel for LSMAS, which, coupled with graph grammars and logic, is suitable for modelling organisational dynamics, especially in the domain of massively multi-player online role-playing games (MMORPGs).

Keywords. organisation, modelling, multiagent systems, large-scale multiagent systems, MMORPG, computer game, dynamics, ontology

Abstract in Croatian

Najpoznatiji i najučestaliji oblici uređenja sustava agenata, velikog ili malog razmjera, su oni koji se temelje na inteligenciji roja i oni koji svoje osnove vuku iz organizacijskih modela. Organizacijski modeli višeagentnih sustava razvijaju se usporedno s ulogom takvih sustava u modernom svijetu. Razvojem tehnologije stvaraju se sustavi koji broje tisuće ili milijune agenata – višeagentni sustavi velikih razmjera (VASVR). Mnogobrojne aplikacijske domene za VASVR (Internet svega, mrežne računalne igre namijenjene većem broju igrača (MMORPG), pametni gradovi i sl.) čine VASVR realno potrebnim konceptom u moderno doba. Recentna istraživanja ukazuju na veću učinkovitost VASVR uređenih temeljem organizacijske teorije, od onih koji prate inteligencija roja, te je stoga organizacijsko modeliranje VASVR iznimno interesantno područje za istraživanje. Organizacijski model temeljen na ontologiji organizacijskih koncepata i modernim načelima organizacije VASVR korak je prema lakšem oblikovanju VASVR. Ontologija je baza za organizacijski metamodel za VASVR koji, spojen s gramatikama grafova i logikom, dobiva na prikladnosti za modeliranje organizacijske dinamike, naročito u domeni MMORPG.

Ključne riječi. organizacija, modeliranje, višeagentni sustav, višeagentni sustav velikih razmjera, MMORPG, računalne igre, dinamika, ontologija

Contents

List of Used Acronyms

ABM	agent-based modelling
ACMAS	agent-centred multiagent system
AMAS	adaptive multiagent system
API	application programming interface
AToM^3	A Tool for Multi-formalism and Meta-Modelling
BDI	belief-desire-intention
DD	data dictionary
DPO	double pushout
GT	glossary of terms
HMAS	holonic multiagent system
ICT	Information and communication technology
IoE	Internet of Everything
IoT	Internet of Things
IVE	intelligent virtual environment
JaCalIVE	Jason Cartago implemented intelligent virtual environment
KB	knowledge base
LPS	Logical Production System
LSMAS	large-scale multiagent system
MAM5	Multi-Agent Model For intelligent virtual environments
MAS	multiagent system
MMOG	massively multi-player online game
MMORPG	massively multi-player online role-playing game
ModelMMORPG	Large-Scale Multi-Agent Modelling of Massively On-Line Role-Playing Games

NPC non-player character

OCMAS organisation-centred multiagent system

OOVASIS cro. Organizacijsko oblikovanje višegentnih sustava u Internetu Stvari - eng. Organizational Design of Multi-Agent Systems in the Internet of Things

OWL Web Ontology Language

RDFS Resource Description Framework Schema

RDF Resource Description Framework

RPG role-playing game

SPADE Smart Python Agent Development Environment

SSSHS Smart Self-Sustainable Human Settlement

TMW The Mana World

W3C World Wide Web Consortium

List of Figures

List of Tables

Listings

Prošireni sažetak na hrvatskom jeziku

Uvod i pregled dosadašnjih istraživanja

Višeagentni sustav (VAS) sastoji se od većeg broja individualnih autonomnih softverskih agenata, čije ponašanje može biti ograničeno određenim skupom pravila, tj. organizirano. Takvi se agenti u svom sustavu nalaze unutar određene okoline na koju mogu utjecati svojim aktuatorima ili iz koje mogu dobivati podražaje korištenjem senzora [116]. Osim okoline u kojoj se nalaze, agenti mogu percipirati druge agente koji se nalaze u okolini istog sustava, što postavlja temelje za njihovu međusobnu komunikaciju, suradnju i organizaciju.

Motivacija ovog istraživanja proizlazi iz uočavanja aplikacijskih domena VAS te mjesta za napredak u domeni organizacijskih modela za modeliranje organizacije u višeagentnim sustavima velikih razmjera (VASVR). Recentna istraživanja koja se bave organizacijom u VAS postavljaju nove standarde za organizacijske modele VAS koji su primjereniji za moderni svijet u kojem raste popularnost tzv. Interneta svega (eng. Internet of Everything (IoE)) i Interneta stvari (eng. Internet of Things (IoT)), a koji uvjetuje rad u rastuće turbulentnoj i kompleksnoj sredini. IoE i IoT [7, 84, 142, 122] su, u svom obliku sustava sastavljenih od raznih objekata, podataka, ljudi i procesa, povezanih konceptima informacijskih i tehničkih znanosti, čime ostvaruju sadržajno bogatiju okolinu nego ikad prije, prepoznati kao odlično prikazivi koncepti apstrahiranjem u VASVR. Uz navedeno, VASVR dodatno podržavaju i razna područja primjene IoE i IoT, od pametne infrastrukture i prometa, do pametnih gradova [137, 135, 139, 144], i mnogih drugih oblika distribuiranih sustava. Naime, upravo osnovne pretpostavke IoE i IoT predviđaju inteligentne sustave koji odgovaraju konceptima distribuiranih i autonomnih sustava – obilježja distribuiranosti, autonomnosti i inteligentnosti jasno opisuju višeagentne sustave.

Unatoč specifičnosti poput projektiranja ontologije te stvaranja organizacijskog metamodela, interdisciplinarnost ovog istraživanja vidljiva je u isprepletenosti područja informacijskih znanosti (višeagentni sustavi velikih razmjera [152, 126]) s ekonomskim disciplinama (modeliranje organizacije [87, 83, 20, 30]), što povezuje ovo istraživanje s mnogim objavljenim istraživanjima, dio kojih je naveden u poglavlju 1.4. Povećana kompleksnost i raširenost VAS te njihova uloga u životu modernog čovjeka dovela je do potrebe za proučavanjem organizacije u takvim sustavima, a s ciljem korištenja brojnosti agenata te njihove suradnje prema zajedničkom cilju i rješavanju prepreka vidljivih u ograničenosti individualnih agenata [3, 4, 62].

Pojam organizacije se temeljno promatra iz dvije perspektive: kao entitet ili kao proces, koje nisu međusobno isključive [31]. Sukladno tome, Abbas, Shaheen i Amin [3] opisuje dva osnovna odnosa prema organizaciji u proučavanju VAS: višeagentni sustavi usredotočeni na agenta (eng. agent-centred multiagent system (ACMAS)), i višeagentni sustavi usredotočeni na organizaciju (eng. organisation-centred multiagent system (OCMAS)). ACMAS se vodi idejom da ne postoji organizacija koja je nametnuta sustavu, već individualni agenti svojim djelovanjem, postupcima i ponašanjem utječu na svoju okolinu te organizacija nastaje temeljem interakcije svakog individualnog agenta s njegovom okolinom. Ovakav pogled na VAS podsjeća na mrave ili roj insekata, no ne predstavlja zadovoljavajuću podlogu za kompleksnije sustave (npr. potencijalno otežava usklađivanje agenata prema zajedničkom cilju) [3, 154]. S druge strane, OCMAS odnos organizaciju promatra u smislu strukture sustava koja je nametnuta agentima. Agenti su upoznati s organizacijom i mogu je mijenjati ukoliko je potrebna prilagodba sustava nestabilnoj okolini. Takvi sustavi mogu imati jasno određen tok informacija i određivanja, kao i komunikacijski protokol, što ih čini podobnijima za kompleksnije sustave. Upravo je sinteza obaju pogleda pogodna za moderne sustave koji djeluju u visoko turbulentnoj okolini [27, 36], kako bi se spojile dobrobiti oba načina organizacije VAS. Nadalje, novija istraživanja, navedena u poglavlju 1.4, često govore o samo-organizacijskim sustavima.

Organizacijski modeli za VAS služe za prikaz arhitekture organizacije agenata, a izražavaju se jezicima za modeliranje sastavljenim od specifičnih simbola. Uobičajeno je da jezik za modeliranje ima dva osnovna elementa [5, 54, 59]: konceptualizaciju (skup koncepata za modeliranje) i sintaksu (pravila za povezivanje elemenata konceptualizacije). Dodatna razmatranja meta- i modeliranja iznesena su u poglavlju 2.2. Prema tome, model [29] je instanciranje sintaktički iskazane koceptualizacije koja opisuje dani sustav. Jezik za modeliranje opisan je metamodelom, tj. modelom modela. Specifična vrsta metamodela je domenska ontologija [54] – konceptualizacija dane domene bez obzira na jezičnu sintaksu.

Coutinho, Sichman i Boissier [29] procjenjuju organizacijske modele za VAS temeljem skupa dimenzija: organizacijska struktura, organizacijske funkcije, organizacijska interakcija, organizacijske norme, organizacijska procjena, organizacijska evolucija, organizacijska okolina, organizacijske ontologije. Od 11 organizacijskih modela analiziranih u navedenom istraživanju, većina se, jasno uočljivo, usredotočuje na modeliranje strukture sustava, dok su organizacijska interakcija te organizacijske funkcije i organizacijske norme sekundarni koncepti modeliranja. Te četiri glavne dimenzije najzastupljenije su, iako su rijetki modeli koji podržavaju modeliranje sve četiri dimenzije (npr. OperA i MAS-ML).

Ostale četiri dimenzije su dodatne, te su rijetko podržane među analiziranim modelima.

U novijem je istraživanju [118, 122] predložen skup perspektiva za procjenu organizacijske arhitekture VASVR, tj. perspektiva koje prema autoru navedenog istraživanja uvelike doprinose učinkovitom djelovanju VASVR. Navedeni skup sastoji se od sljedećih sedam perspektiva: organizacijska strukture (tok informacija i odlučivanja unutar organizacije), organizacijska kultura (nefizički elementi organizacije poput znanja, normi, jezika i slično), strategija (dugoročni ciljevi organizacije, planovi za njihovo postizanje te načini mjerenja uspjeha), procesi (aktivnosti organizacije), individualni agenti (osnovne pokretačke jedinice organizacije), organizacijska dinamika (organizacijske promjene i reorganizacija), te kontekst i međuorganizacijski aspekti (organizacijsko ponašanje prema okolini).

Osim navedenog, moderna istraživanja [72, 3] konačno uvode i temporalno-dinamičnu komponentu organizacije u organizacijske modele argumentirajući modele za VAS u realnom vremenu i promovirajući reorganizaciju VAS. Dodatna diskusija o modelima VAS nalazi se u poglavlju 1.4.

Ciljevi istraživanja

Osnovno istraživačko pitanje je: od kojih se elemenata sastoji skup koncepata primjenjivih na organizacijsko modeliranje VASVR, s naglaskom na organizacijsku dinamiku, i na koji su način oni primjenjivi?

Glavni cilj ovog istraživanja, temeljem navedenog osnovnog istraživačkog pitanja, definiranje je ontologije koja obuhvaća bitne odabrane organizacijske koncepte vezane uz VASVR te na njoj temeljenog organizacijskog modela za VASVR koji poštuje moderne perspektive organizacijskog modeliranja VASVR, s naglaskom na organizacijsku dinamiku.

Za potrebe usmjeravanja istraživanja te provjere uspješnosti dobivenih rezultata, a temeljem glavnog cilja istraživanja, definirano je nekoliko istraživačkih ciljeva:

- C1 Istražiti koncepte organizacijskog modeliranja koji su pogodni za modeliranje organizacije u VASVR.
- C2 Modelirati organizacijske koncepte primjenjive na MMORPG.
- C3 Istražiti modeliranje organizacijske dinamike u primjeni VASVR na MMORPG.

Metodologija

Istraživanje je jasno podijeljeno u tri elementa, koji odgovaraju ciljevima istraživanja: ontologija, metamodel, i procjena.

S obzirom na ulogu temelja rezultata ovog istraživanja, temeljita ontologija koja obuhvaća odabrane bitne elemente organizacijskog modeliranja VASVR ključan je element. Stoga je važno odabrati dobru metodologiju za inženjering ontologija. Ovaj dio istraživanja predviđa ukupno šest koraka, prema METHONTOLOGY, koja je odabrane među metodologijama za inženjering ontologija predstavljenom u [63]. Postupak izrade ontologije opisan je u poglavlju 2.1.

Prvi korak odabrane metodologije je specifikacija koja rezultira specifikacijskim dokumentom zapisanim prirodnim jezikom. Namjena ove ontologije je okupljanje i obuhvaćanje odabranih organizacijskih koncepata relevantnih za organizacijsko modeliranje VASVR. Ontologija smije obuhvaćati i specifične i općenite koncepte, jer će za stvaranje metamodela biti odabrani samo najbitniji identificirani elementi. Koncepti ontologije bit će zapisani korištenjem OWL (eng. Web Ontology Language) jezika. Drugi korak je akvizicija znanja, a provodi se usporedno s definiranjem specifikacije. Ovo je nezavisna aktivnost koja gubi intenzitet napredovanjem procesa definiranja ontologije, ali uvelike ovisi o cilju istraživanja te njegovoj dekompoziciji. Glavni izvor podataka za ovaj dio istraživanja istraživanje je objavljeno [126, 118] u sklopu OOVASIS[1] projekta, uz pristanak autora. OOVASIS ontologija nadograđena je tijekom Large-Scale Multi-Agent Modelling of Massively On-Line Role-Playing Games (ModelMMORPG) projekta, no ovdje se za istu tu ontologiju koristi isti naziv – OOVASIS. Sljedeći korak je konceptualizacija, a sastoji se od izrade pojmovnika te grupiranja i opisivanja identificiranih pojmova. Četvrti korak je integracija, tj. povezivanje s metaontologijama i drugim ontologijama, koliko je to moguće i smisleno. Peti korak izrade ontologije je implementacija koja uključuje odabir i implementaciju ontologije u odabranom alatu. Ontologija će većinom biti definirana korištenjem Protégé alata zbog njegovog *de facto* standarda u semantičkom modeliranju u akademskom kontekstu. Zadnji korak je evaluacija, tj. tehnička procjena ontologije, zajedno s njenom softverskom okolinom i dokumentacijom, u odnosu na referentnu stavku, koja može biti dokumentacija proizašla iz prvog koraka. Ovaj korak podrazumijeva i verifikaciju i validaciju ontologije.

Pregledom metodologija za inženjering ontologija koji su objavili Iqbal i dr. [63], zajedno s jasnim značajkama koje nudi svaka od ontologija te potrebama autora, odabrana je metodologija METHONTOLOGY [42]. Ova metodologija odabrana je zato što nudi razvoj korištenjem razvojnog prototipa, daje podršku za ponovnu primjenu rezultata, nije ovisna o specifičnog aplikaciji, daje jasan prijedlog životnog ciklusa ontologije te je detaljno opisana. Navedeni alat (Protégé) odabran je zbog svoje popularnosti u predmetnoj akademskoj i stručnoj zajednici te izradi ontologija, kao i prirode otvorenog koda.

Nakon definirane ontologije slijedi dio istraživanja koji se bavi stvaranjem metamodela, predstavljen u poglavlju 2.2. Znanstvena metoda modeliranja sastoji se od četiri faze [Žugaj, 2007]: postavljanje zadatka, izbor ili stvaranje modela, istraživanje modela

[1]Više informacija dostupno na `http://ai.foi.hr/oovasis`

te prijenos spoznaja s modela na original. Shodno tome, a uzimajući u obzir i ostatak diskusije u poglavlju 2.2, identificirano je nekoliko koraka u postupku izrade organizacijskog modela. Prvi korak je određivanje detaljnosti, a zahtijeva određivanje razine specifičnosti metamodela te dubine modeliranih koncepata, tj. razinu apstrakcije promatrane domene. Suviše apstraktni koncepti dovode do neizražajnosti modela, dok prevelika konkretizacija domenskih koncepata stvara potencijalno previše kompleksan model. Obje krajnosi su, dakako, loše, te mogu dovesti do otežane primjenjivosti modela. Drugi korak je ocjena i odabir elemenata, u što je uključena analiza ontologije te procjena korisnosti ili iskoristivosti uključenih koncepata. Ontologija organizacijskih koncepata primjenjivih na VASVR potencijalno obuhvaća koncepte koji nisu prilagođeni uključivanju u metamodel ili su dovoljno neutjecajni na konačni ishod da mogu biti isključeni iz finalne verzije metamodela. Ovaj korak rezultira popisom koncepata odabranih za uključivanje u modelirani metamodel. Usporedba i informiranje treći su korak, koji obuhvaća analiziranje, procjenu i usporedbu postojećih VAS organizacijskih modela i njihovih koncepata. Cilj ovog koraka je uočiti dobre primjere koji odgovaraju krajnjem cilju ovog istraživanja te ih prilagoditi za razvijani metamodel. Četvrti korak je samo stvaranje metamodela. Usporedbom koncepata odabranih u trećem koraku i onih uočenih u četvrtom koraku te njihovim spajanjem, razvija se metamodel. Slijedi uključivanje odabranih koncepata u metamodel pomoću odabranog alata AToM^3. Zadnji korak ovog dijela istraživanja usporedba je metamodela sa sedam perspektiva i evaluacija modela temeljem istih. Ovaj korak iznimno je bitan zbog ostvarivanja svojevrsne povratne veze te stvaranja ocjene razvijenog metamodela. Modelirani model te njegove instance bit će uspoređeni sa sedam perspektiva organizacijske arhitekture VASVR [118]. U slučaju nedovoljnog zadovoljavanja zadanih kriterija i obilježja navedenih perspektiva, potrebno je ponoviti slijed od četvrtog ili drugog koraka.

Navedeni alat (A Tool for Multi-formalism and Meta-Modelling (AToM^3)) odabran je zbog svoje prirode otvorenog koda, iznimno dobre povezanosti s programskim jezikom Python, te zadane namjenjenosti procesu metamodeliranja. Navedeni alat omogućava grafičko stvaranje modela te njegovo korištenje i prilagodbu raznih aspekata tog modela. Povezanost s programskim jezikom Python bitna je zbog jasne i efikasne za korištenje platforme za stvaranje višeagentnih sustava, SPADE (eng. Smart Python Agent Development Environment). SPADE je jedinstven po tome što je prvi potpuno temeljen na XMPP tehnologiji. Dodatna evaluacija SPADE-a iznesena je u poglavlju 3.1.

Organizacijski metamodel dobiven tijekom ovog istraživanja procijenjen je kako slijedi. Usporedbom s nekim od vodećih postojećih i razvijenih organizacijskih modela utvrđene su prednosti i nedostaci razvijenog modela. Primjenjivost modela ovog istraživanja procijenjena je temeljem njegove primjenjivosti prvenstveno na aplikacijsku domenu u vidu mrežne računalne igre s ulogama namijenjene većem broju igrača (eng. massively multiplayer online role-playing game, MMORPG), a zatim i dodatne dvije domene, jednu bližu kontekstu modeliranja temeljenog na agentima, i drugu koja svoju primjenu nalazi u kontekstu Interneta stvari i pametnih gradova. MMORPG igre prepoznate su kao jedan od odličnih primjera primjene VASVR, dok je specifična igra The Mana World odabrana za okolinu testnog scenarija zbog svoje prirode otvorenog koda, besplatnog sudjelovanja u igri, jednostavnosti uređivanja raznih aspekata virtualnog svijeta, sadržavanja koncepata često korištenih u domeni MMORPG igara, te njenog korištenja u ModelMMORPG projektu dio kojeg je i ova disertacija.

Disertacija je strukturirana kako slijedi. Poglavlje 1 opisuje motivaciju istraživanja, osnovne definicije korištenih pojmova, s ciljem lakšeg snalaženja i razumijevanja ostatka sadržaja, te pregled povezanih istraživanja. Znanstveni doprinos opisan je u poglavljima 2.1 i 2.2, gdje su zasebno izneseni detalji procesa semantičkog modeliranja i metamodeliranja. Praktični doprinos ovog istraživanja predstavljen je u poglavlju 3.1. Primjeri primjene razvijenog modela opisani su u poglavlju 4, dok je zadnje poglavlje rezervirano za diskusiju o iznesenom sadržaju. Sadržaj dodataka služi za pobliže određivanje ili pojašnjenje određenih tema disertacije te su prema tome referencirani u sadržaju disertacije.

Chapter 1

Introductory Notes and Related Research

1.1 Motivation

This research is motivated by observing application domains of multiagent systems (MASs) and advancement possibilities in the domain of organisational models for large-scale multiagent systems (LSMASs). Recent studies on organisation in MASs suggest new standards for organisational models for MASs that are more suitable for the increasingly turbulent and complex modern world where the Internet of Everything (IoE) and the Internet of Things (IoT) are growing in popularity. IoE [7, 84, 142, 122] is, as a specific combination of various objects, data, people and processes creating environment contextually richer than ever before, recognised as a concept appropriately described and abstracted using LSMASs. Furthermore, confirmation of the stated can be found in application domains of IoE, e.g. smart infrastructure, smart transportation, smart cities, etc. [137, 135, 139, 144, 65] The specific basic features of IoE demand and assume intelligent systems that conform to concepts of distributed and autonomous systems. Features of such systems, like distribution, autonomy, and intelligence clearly represent multiagent systems.

Another key domain that is gaining popularity in research related to MASs are computer games. A specific genre of computer games is in the spotlight of this research: massively multi-player online role-playing games (MMORPGs), which are a combination of the genre of massively multi-player online games (MMOGs) and role-playing games (RPGs). The interest in conducting research in the domain of RPGs is based on personal attraction to such a genre of computer games, and the observed applicability of RPG dynamics and mechanics to the domain of MASs. Coupled with the genre of MMOGs, which feature large numbers of players who interact with each other and the in-game world, MMORPGs represent a highly interesting field of research in the context of MASs. Quest driven gameplay, vast possibilities in the context of available player actions, and encouraged or demanded social interaction of player agents, are only some of the features of MMORPGs that make them a good research topic when MASs and LSMASs are concerned.

Although this research contains specific elements such as ontology engineering, and defining an organisational metamodel, interdisciplinary nature of the research visible in interwoven areas such as information sciences (LSMASs [152, 118]), and economical disciplines (organisational modelling [87]), clearly relates this research to many published studies. Increased complexity and presence of MASs, along with their meaning in a life of the modern human, led to the necessary research of organisation in such systems, aimed at utilising benefits of agent numbers, their cooperation towards a common goal, and individual agent's constraints [3, 4, 62].

1.2 Introduction

A MAS consists of a great number of individual autonomous software agents. Their behaviour can be constrained using a set of rules, i.e. organised. Such agents are located within an environment they can act upon using their actuators, or get feedback from using their sensors [116]. In addition to interacting with their environment, agents can perceive other agents in the system's environment, thus forming the basis for implementing organisational features through communication and cooperation with each other.

The concept of organisation is usually observed from two perspectives: as an entity, or as a process, but not necessarily mutually excluded [31]. Both are closely related to later studies of Abbas, Shaheen and Amin [3], who recognise two basic approaches to how the concept of organisation is observed in research on MASs: agent-centred multiagent systems (ACMASs), and organisation-centred multiagent systems (OCMASs).

An ACMAS delves on the idea that there is no organisation that would be cast upon the system by design. The concept of organisation in an ACMAS is built in the bottom-up manner and it emerges from agents affecting each other and interacting amongst themselves and with their environment. Behaviour, actions, and interaction of agents are thus said to produce organisation as a concept in an ACMAS [19]. Such a perspective on MASs reminds of ants and insect swarms, yet it alone is not presented as a beneficial solution for complex systems [3, 154, 12]. Depending on the intended nature of the observed system, the ACMAS approach can render the whole system unresponsive to its rapidly changing environment or otherwise inhibit its performance as a result of the emergent nature of the behaviour of the whole system. Furthermore, one of the arguments against clean ACMAS implementation are overburdened agents that are given the responsibility of system organisation in addition to their regular functional responsibilities [154].

An OCMAS, on the other hand, considers organisation as a concept enforced upon the system and the included agents. Agents are aware of the organisation though, and they can influence on it when they recognise that the system should be adapted to the unstable environment. Such systems usually have a clearly defined information and decision flows, as well as communication protocols. Therefore, an OCMAS approach is more suitable for complex systems, for it usually allows the system to produce a response to unpredictable situations faster than an ACMAS system. Clearly, both the argument for ACMAS, and that for OCMAS, depend on the intended purpose of the observed system.

In the end, it is actually the joint view on organisation that which is the most beneficial for the modern systems functioning in highly turbulent environments [27, 36]. Such an approach would assure benefits of the both described perspectives.

Coutinho, Sichman and Boissier [29] evaluate organisational models for MASs using the following set of dimensions, of which four are basic, and four are additional dimensions, respectively:

- organisational structure,
- organisational functions,
- organisational interaction,
- organisational norms,
- organisational evaluation,
- organisational evolution,
- organisational environment,
- organisational ontologies.

Out of the eleven organisational models analysed in [29], most of them feature the concepts of organisational structure, yet only a smaller number of models comprise concepts of organisational interaction, organisational functions, and organisational norms. It is expected that the mentioned four basic dimensions are present in most of the analysed models, whereas the four additional dimensions are often missing. However, two models contain concepts of all the basic dimensions (OperA and MAS-ML). Some of these models are detailed in Section 1.4.3.

A newer research, conducted by Schatten et al. [126] and Schatten, Ševa and Tomičić [122], presents a revised set of organisational modelling perspectives that are argued to aid more in building efficient LSMASs constrained by organisational features. The mentioned set contains the following seven perspectives:

- organisational structure (decision and information flows of an organisation),
- organisational culture (important intangible aspects of an organisation including knowledge, norms, reward systems, language and similar),
- strategy (long term objectives of an organisation, action plans for their realisation as well as tools on how to measure success),
- processes (activities and procedures of an organisation),

- individual agents (the most important asset of any organisation – individual agents actually performing the work),
- organisational dynamics (organisational changes including reorganisation of any of the mentioned components),
- context and inter-organisational aspects (organisational behaviour towards its environment including strategic alliances, joint ventures, mergers, splits, spinouts, and similar).

Furthermore, recent studies [72, 3] finally introduced a temporally-dynamical organisational component to LSMAS organisational models arguing the need for real-time LSMAS models and promoting reorganisation in LSMASs.

MMOGs are an interesting application domain of LSMASs for several reasons. The rising popularity complex computer games gained with the development of digital infrastructure led to an increased importance of various aspects of how computer systems are used, and to what ends, and computer games (including consoles) are one of them. Greater technology availability motivated the gaming industry to invest more into game development, thus creating a whole new domain of Information and communication technology (ICT) in its own right. Computer games have since become a lucrative business, with market shares measured in dozens of billions of USD. [123] In this case, huge market share is caused by a large user base, meaning that video games have a high popularity. The trend that is growing for a couple of years is the development of MMOGs – video games that are designed for a large number of players playing the game simultaneously, possibly interacting with each other, and the in-game world. MMOGs represent a whole new medium that can be used for social interaction of end-users, and a self-realisation tool in the online world. Out of many of the video game genre that adapted to the MMOG environment, the most interesting genre for this research is MMORPG because games of this genre explicitly make their players assume a role and exercise its abilities and other features in an in-game world.

MMORPGs therefore face their players with a virtual world accessible to their avatars in this world, and confronts them with various challenges ranging from simple tasks to complex campaigns. The in-game world is rich in player avatars (hence the MMOG genre), non-player characters (NPCs), and numerous other creatures that inhabit the given virtual world. Players are usually presented with a set of quests that can be dealt with in a linear or a non-linear manner. Modern games leave this choice to individual players, as the idea of an open world (a world without strict story-bound constraints) is prevalent. One of the key aspects of an MMORPG is interaction – social interaction between player avatars and other creatures of the given world, as well as interaction of player avatars with the world in general. The social component is by large what makes MMORPGs very interesting for research in modern systems comprising artificial agents, as well as for this particular research. Namely, individual players can advance through an MMORPG, yet their progress grows slower as they advance through the game. As the game advances, players can gain increased benefit from interacting with other players (in games that stimulate cooperative gameplay), and forming various types of groups of players (parties, and guilds, as described earlier here). Such coalitions or groups or organisations help individual players best the challenges they are faced with through the game. The nature of such groups, and their purpose, varies between games, with the standardised notion of a party as a temporary quest-centred grouping mechanism, and a guild as a longer-lasting grouping mechanism with emphasised social components. Furthermore, some in-game challenges are designed for larger numbers of organised players with a tactful approach.

MMORPGs usually have players playing characters belonging to a single, a pair of, or a number of character classes – usually using stereotyped character descriptors – warriors, archers, thieves, wizards, druids, etc. Depending on the class the character plays, different parts of the game are usable to the player, including varying gear, abilities, interactions, etc. MMORPGs are usually computer games that are quest-driven, i.e. game dynamics in the context of a story and campaign and game advancement is governed by in-game quests usually obtainable through interaction with NPCs or special in-game events. These quests yield special rewards for their completion (e.g. special kind of loot, new quests, etc.). Some quests depend on the player's character being able to perform a specific in-game action or interaction, thus underlining the importance of character actions. The described view on the MMORPGs domain can be simplified and represented using the Lamrast−+ metamodel, in order to create an artefact that can be further used in the modelled system's development. Research on social interaction of players, and its replication on artificial agents, with the goal of creating agents that are able to cooperatively solve more complex quests, is one of the integral parts of Large-Scale Multi-Agent Modelling of Massively On-Line Role-Playing Games (ModelMMORPG) project, a part of which is this thesis.

The specific game of the MMORPG genre that is used in this research, and was the main research environment of ModelMMORPG project, is The Mana World (TMW)[1] – an open-source 2D MMORPG. Although it looks simple, TMW includes all the key elements of an MMORPG, including avatar customisation, social interaction, quests, avatar grouping features, etc. Furthermore, TMW can be modified and customised as necessary, which was important for this particular research, as a specific quest was developed for the purpose of collecting initial research data.

A welcome addition to the scientific contribution and theoretical part of this research is the modelling tool that is built upon the defined Lamrast−+ metamodel. This practical contribution is what sets this research apart from most of the research into modelling LSMASs, since a lot of the already published models (see Section 1.4.3 for further discus-

[1]For more information visit `https://www.themanaworld.org`

sion) remain at theoretical-only level. However, Lamrast−+ metamodel is supported by its proprietary modelling tool that is built as a customisation of the open-source metamodelling tool A Tool for Multi-formalism and Meta-Modelling (AToM^3). The most prominent feature of this customised metamodelling tool is application template generation based on the defined model of a system comprising agents. Additionally, the modelling tool provides necessary mechanisms for using graph grammars, which are used in this research for the purposes of modelling organisational dynamics (especially evident in MMORPGs considering the existence of both parties and guilds). Further described in Section 3.3.2, application template generating feature of the customised metamodelling tool helps system developers receive basic implementation template of the modelled system.

1.2.1 Research Objectives

The basic research question of this thesis is: What are the elements included in the set of concepts applicable to organisational modelling of LSMASs, emphasising organisational dynamics, and how can they be used?

The main objective of this research, based on the research question, consists of the following: defining an ontology comprising chosen organisational concepts applicable to LSMASs, and an organisational metamodel for LSMASs based on the mentioned ontology, conforming to the modern perspectives of organisation modelling, emphasising organisational dynamics.

With the idea of guidance and evaluation support, several research objectives are defined based on the main objective:

O1 Analyse organisational modelling concepts applicable to LSMASs.

O2 Model organisational concepts applicable to MMORPGs.

O3 Explore modelling of organisational dynamics in MMORPGs as a specific application of LSMASs.

1.2.2 Initial Research Plan

The research covered by this thesis is divided into three parts: an ontology, a model, and evaluation.

Considering its key role in this research, adequate ontology comprising selected organisational concepts applicable to LSMASs is fundamental to the research results. Therefore, a good ontology engineering methodology is needed. Six steps are identified in this part of research, based on the chosen ontology engineering methodology, METHONTOLOGY, whose selection motivation is presented in Section 2.1.1. Result of the first step (specification, detailed in Section 2.1.1.1) is a specification document written in natural language that contains basic information about the ontology being developed. Purpose of this ontology is to collect the chosen organisational concepts applicable to LSMASs. Ontology concepts will be written in Web Ontology Language (OWL). The second step (knowledge acquisition, detailed in Section 2.1.1.2) is performed simultaneously with the specification step. This independent activity weakens in intensity as the ontology definition process advances, and is greatly influenced by the main research goal. This step will mostly rely on the research conducted during cro. Organizacijsko oblikovanje višegentnih sustava u Internetu Stvari - eng. Organizational Design of Multi-Agent Systems in the Internet of Things (OOVASIS) project [118, 126]. The third step (conceptualisation, detailed in Section 2.1.1.3) is about building an index of the chosen and identified concepts with their definitions. The fourth step (integration, detailed in Section 2.1.1.4) connects the developed ontology to metaontologies and other ontologies, where sensible and possible. The fifth step (implementation, detailed in Section 2.1.1.5) works on coding the developed ontology using a chosen tool. The ontology will mostly be defined using Protégé. The last step of METHONTOLOGY (evaluation, detailed in Section 2.1.1.6) serves to carry out a technical judgement of the developed ontology, including the accompanying software environment and documentation, with respect to a frame of reference (e.g. documentation from step one). This step includes ontology verification and validation.

METHONTOLOGY [42] ontology engineering methodology was chosen based on the review of ontology engineering methodologies published by Iqbal et al. [63]. The chosen methodology is based on a developing prototype, has reusability support, is not dependent on a specific application environment, is very well described, and has a clear ontology life cycle recommendation. Protégé was chosen because it is open-source, very popular in academic and real sectors, and is widely accepted tool for ontology engineering. OWL was chosen as an ontology language for its role of a *de facto* standard in the domain of semantic modelling.

After the ontology is defined, the associated model is to be developed. The scientific model developing method consists of several phases [157]: setting a goal, building a model, detailing the model, and applying the model. According to these phases, five steps were identified, as follows. The first step (detailed in Section 2.2.1.1) is about choosing the level of abstraction of the model, i.e. in how much detail will the final model be modelling the given domain. Clearly, very abstract concepts will not give an expressive model. Contrariwise, very specific concepts may make the final model hard to use. Both extremes are undesirable, since the final model may hardly be usable. The second step (detailed in Section 2.2.1.2) is about choosing concepts for the model being built, by analysing usefulness and usability of all the concepts of the defined ontology. The ontology may comprise concepts that are not suitable for the developing model, and should therefore not be included, e.g. they are not important enough. Result of this step is a list of concepts that will be a part of the final model. Third step (detailed in Section 2.2.1.3) is about analysing, assessing, and comparing concepts and LSMAS organisational models that already exist. Additionally, analysis will be conducted on more general elements, e.g. normative systems. The fourth step (detailed in Section 2.2.1.4) is developing the actual model. Organisational dynamics (detailed in Section 2.2.2) will be defined and described using, but not limited to, graph grammars (definitions provided in Appendix B.2), and temporal logics. The chosen concepts are integrated in a model using AToM^3. The last step (detailed in Section 2.2.1.5) of this part of the research is assessment of the metamodel, based on the seven perspectives, and its evaluation. The defined metamodel, and its respective instances, will be compared to the seven perspectives of organisation architecture for LSMAS [118]. In case the criteria will not be met, the development process should be repeated, and the metamodel improved.

AToM^3 is chosen for this research since it is an open-source software working with Python, and is by default built for meta- and modelling purposes. Furthermore, it makes it possible to graphically create and use a model, and to introduce numerous modifications to it. Easy integration with Python is important since a clear and efficient platform for development of MAS – Smart Python Agent Development Environment (SPADE) – is developed using Python, and Python provides many libraries of various possible applications. SPADE is the first such piece of software to use a particular popular communication protocol (XMPP).

Organisational metamodel developed during this research is evaluated as follows. Firstly, the developed metamodel is compared to some of the leading already existing organisational models on conceptual level, and similarities and differences are noted. Secondly, applicability of the model is tested using an MMORPG as a good example of LSMASs application domain, with further testing conducted on an example closer to the agent-based modelling (ABM) context, and another, applicable to IoT and smart cities. The tests were conducted on three testbed scenarios. In addition to being an MMORPG, and therefore being recognised as a decent LSMASs application example, the specific game TMW is chosen as the test environment for its open-source nature, free game participation, easy modifications of the in-game world, inclusion of an average number of elements from the MMORPG domain, and because it is used in ModelMMORPG project, a part of which is this thesis.

This research is set to develop basics for easier modelling of LSMASs in the context of MMORPGs. Scientific contribution is recognised in the ontology comprising organisational concepts applicable to LSMASs, and a organisational model for LSMASs (applied to MMORPGs) based on the mentioned ontology and modern features of organisational modelling for LSMASs, emphasising organisational dynamics. In addition to the scientific contribution, a practical contribution is presented as well, in form of an application template generating tool for basic parts of LSMASs modelled using the provided modelling tool, which supports use of graph grammars as well.

1.3 Conceptual Definitions

The following mostly short definitions are used to clearly define the scope of various concepts used further in this thesis. The purpose of this is to avoid confusion and to clearly state the scope of each of the key concepts of this thesis. Intensions of various concepts will be defined here, although further discussions about a concept may be present in other parts of this thesis. The following concepts are grouped by their meaning and value for this thesis, and are not presented in alphabetical or similar order.

agent *An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.* [116]

multiagent system (MAS) *Multiagent systems are the best way to characterize or design distributed computing systems.* [151] Key characteristics of MASs are: infrastructure specifying communication and interaction protocols, open environment with no centralised designer, and autonomous, intelligent, distributed agents that behave cooperatively or out of self-interest.

large-scale multiagent system (LSMAS) MASs that comprise a large number of agents, and base their complexity therein.

model Abstract approximative representation of a real domain. *A model is a representation of something for someone's purpose somebody (sic) and developed by someone else.* [134].

metamodel Shortly defined, a metamodel is a model of a model.

ontology The context of the concept of ontology is constrained within this thesis to the domain of information and computer sciences. *An* ontology *is an explicit specification of a shared conceptualization that holds in a particular context.* [52]

organisation *Organizations are (1) social entities that (2) are goal-directed, (3) are designed as deliberately structured and coordinated activity systems, and (4) are linked to the external environment.* [30]

1.4 Related Research

The following section presents the published research related to the concepts and the context of this thesis. For the sake of clarity, the contents of this section are divided into three parts: one that contains research related to organisation in the context of MASs (Section 1.4.1), followed by the part about semantic modelling and the use of ontologies in the domain of MASs and LSMASs (Section 1.4.2), with the part on the use of models in the domain of MASs and LSMASs at the end of this section (Section 1.4.3).

1.4.1 The Concept of Organisation in Multiagent Systems

Organisation as a concept in systems comprising artificial agents, especially those of a large scale, is a matter of an ever growing body of research. With the development of the IoT paradigm, along with the research towards smart cities [74], smart grid [23, 135, 47, 23], and ultimately the IoE [122], it is important to work on the protocols that foster agent interaction and, even better, their cooperation towards reaching a common goal shared amongst a group of agents, or a whole organisation – an approach completely in accordance with what is stated here above. MASs are developed as self-organising systems as well, with the role of other (complex) system controllers, such as the results described by Boes and Migeon [16], where the adaptive multiagent system (AMAS) approach is used. A similar research on self-organisation of MASs in the form of a swarm, realised using a distributed control system, is described by Krishnan and Martínez [71]. Furthermore, the concept of self-organisation is recognised as a useful feature in smart environments by Cameron et al. [21] as well, since the emerging organisation and coordination mechanisms emerge from behaviour of individual agents, thus rendering the whole system more resilient, when the aspect of a single-point-of-failure is examined. Further examples are available, for example in [133].

Since the core of this research is a combination of semantic and organisational modelling, what should be mentioned here is *a core ontology (ORG) for organizational structures, aimed at supporting linked data publishing of organizational information across a number of domains.* [145] This ontology, named The Organization Ontology, was published by W3C [145] with the intention of creating a basic ontology that is ready for domain-specific extensions which could add more specific classification of the included organisation and roles, all the way to extensions such as organisational activities, or perhaps the concepts discussed further in this thesis, such as normative concepts beyond roles and similar.

Main discussion on various meta- and models of the domain of MASs is located in Section 1.4.3, yet the research which is of fundamental significance for this research should be reported about here as well, since it is related to both organisational modelling and MASs. A research done by Schatten [118] and Schatten, Ševa and Tomičić [122] presents a set of perspectives for organisational modelling that is argued to aid in building efficient LSMASs constrained by organisational features. The mentioned set contains the following seven perspectives: organisational structure (decision and information flows of an organisation), organisational culture (important intangible aspects of an organisation including knowledge, norms, reward systems, language and similar), strategy (long term objectives of an organisation, action plans for their realisation as well as tools on how to measure success), processes (activities and procedures of an organisation), individual agents (the most important asset of any organisation – individual agents actually performing the work), organisational dynamics (organisational changes including reorganisation of any of the mentioned components), as well as context and inter-organisational aspects (organisational behaviour towards its environment including strategic alliances, joint ventures, mergers, splits, spinouts, and similar).

1.4.2 The use of Semantic Modelling in Multiagent Systems

Modelling in general is viewed as a method of creating models. In this context, a model is considered an abstract representations of a real domain. The building blocks of a model – concepts – are constrained by a selected set of properties of real-life concepts [134]. A concept is described using its three main descriptors: intension, extension, and symbol. An intension is basically a definition of the concept, its description using features of the concept that define it for what it is, no more, no less. The extension includes all the instances of the given concept. The symbol is a way of referencing the given concept. Early examples of the use of symbols and concepts to represent human thoughts is described using examples of ancient Egyptian hieroglyphs in [25]. All three concept descriptors are exemplified in Section 2.2.

The process that is used to apply various concepts to objects of the real domain is called classification. Using the concept of extension, classification can be described as the process of populating the extension set of a given concept. Classification has multiple benefits [105]: it can be used as a process of structuring knowledge featuring concepts and their associated objects from the real domain, and it fosters the reasoning process thus rendering exhaustive definition of property values unnecessary, since some of the needed property values can be inferred based on those that are defined. Further discussion on concepts and the notion of modelling is given in Okreša Đurić and Maleković [100, 99].

Since the process of modelling, when conceptual modelling is used, is of high importance, as it represents the basis for all the information and knowledge that will be stored through the knowledge management process, four key elements are identified by Thalheim [134] that characterise a good model: a source (the basic properties necessary for a good description of the modelled source, along with the purpose of the model, goals of its creation and application, the context of the model and the source, etc.), concepts (definitions of the concepts, their applicability, constraints, etc.), a view, and understanding (user profiles, their intentions, and other features). The reasoning around the listed four characteristics of a good model and their relevance to models of knowledge management is rooted not only in the benefits stemming from a well implemented knowledge management practice rooted in innovation, but in the sole knowledge management process as well, especially in its sharing part, and application phase. For the purposes of the aforementioned use of knowledge management (KM), KM is defined as the process of discovering, acquiring, storing, sharing and applying knowledge.

While conceptual modelling is concerned with concepts and their definition through the three descriptors referenced above, semantic modelling is about enriching conceptual models with semantic information. In the context of distributed systems (e.g. MASs and IoT or IoE), application of semantic technologies thus helps interoperability, promotes integration and data transportation, fosters reasoning, and knowledge discovery and extraction Wang et al. [150]. Following the set context, semantic models can be used to assure more detailed semantic annotations for various system elements, including services, resources, data, etc.

The main outcome of a semantic modelling process is an ontology – a knowledge model, in its most basic definition, in the context of information and computer sciences. Schreiber [128], building on Gruber [52], provides a tentative commonly used definition as: *An* ontology *is an explicit specification of a shared conceptualization that holds in a particular context.* An older definition, again in the context of information and computer sciences, is given by Smith [131]: *an ontology is a software (or formal language) artefact designed with a specific set of uses and computational environments in mind.* Russell and Norvig [116] describe an ontology as the result of one of the steps in the knowledge-engineering process using first-order logic, namely deciding on a vocabulary of predicates, functions, and constants. An ontology is thus described as *a particular theory of the nature of being or existence. The ontology defines what kinds of things exist, but does not determine their specific properties and interrelationships.*[116] It should be noted here that the concept of an ontology in the context of information and computer sciences is somewhat different from the same concept in the context of philosophy. Whereas information and computer sciences refer to a piece of software, in the context of philosophy ontology is the branch of metaphysics that deals with the nature of being, and the basic categories of being and their relations[2]. Further in this thesis, the concept ontology is used in the context of information and computer sciences.

By its definition, an ontology consists of interrelated concepts. The included concepts are defined using various constraints, and usually related to other concepts using relations. While it is suitable to talk about relations only on the lower levels of implementing semantics, when using OWL 2 (as is the case of this research), *we denote objects as individuals, categories as classes and relations as properties.* [147] Therefore, while relations are the entities connecting various concepts of an ontology, they are mostly referred to as properties in this thesis. Furthermore, since the Lamrast−+ ontology is defined using OWL 2 constructs, two types of properties can be discerned: object properties and data properties. Object properties are relation entities connecting two concepts or objects of the defined ontology (e.g. two individuals of type `ComicBookCharacter`), while data properties are entities that relate an object to a data type (e.g. an individual of type

[2]Collins English Dictionary - Complete & Unabridged 2012 Digital Edition, ©William Collins Sons & Co. Ltd. 1979, 1986 ©HarperCollins Publishers 1998, 2000, 2003, 2005, 2006, 2007, 2009, 2012

`ComicBookCharacter` and data of type string).

Three main ontology types are defined by Schreiber [128]: foundational ontologies, domain-specific ontologies, and task-specific ontologies. Foundational ontologies (also called upper ontologies [116, p. 437]) stay the truest to their original concept from philosophical studies [131] where an ontology is the theory of that what is, considering they *aim to provide conceptualizations of general notions, such as time, space, events, processes.* [128] Domain-specific ontologies are related to a specific domain, i.e. a specific context, and are intent on providing concepts and their relations in a particular area of interest. One such ontology is the one of this research, presented further in this thesis (Section 2.1), since it is related to a specific domain of LSMASs and MMORPGs. The lowest-level ontology form are task-specific ontologies which provide conceptualisations needed for performing a particular task. A somewhat similar specification is provided by Russell and Norvig [116], which recognises general-purpose, and specific-purpose ontologies.

The shared aspect of an ontology in information and computer sciences, emphasised in the above definitions, is the most interesting, for an ontology is created with the main goal of supporting and promoting knowledge sharing. What is shared is a conceptualisation, i.e. an abstract model of a specific phenomenon, or of specific domain, in terms of concepts and their relations usually in the form of modelled concept properties. These concepts are further specified using various terms and semantic features. Such a specification is presented in a defined, clear, and precise form, using definitions and formalisms made explicit using a language that is, preferably, understandable by both humans and artificial agents.

Ultimately, every ontology, be it a foundational, a domain-specific, or a task-specific one, is related to a particular context. Indeed, context is of great importance when knowledge stored in an ontology is reused (thus fulfilling its main purpose) [128], as it is unreasonable to expect various people or artificial agents to understand an author's conceptualisation, if no context is provided.

A further overview of semantic modelling was conducted in a Master thesis by Okreša Đurić [95]. The referenced work provides an overview of semantic modelling and the languages used for describing an ontology (Resource Description Framework (RDF), Resource Description Framework Schema (RDFS), and OWL), as well as an insight into semantic modelling of business rules.

Even though in their description and basic use, ontologies may be very similar to an ordinary data model, the differences exist, and are situated in the emphasised intent of ontologies – a set of concepts to be shared amongst users (human and artificial) and applications. Ontologies are therefore created with an open world in mind (using the open world assumption) where distributed knowledge is appreciated, while data models are meant for relatively small, but more importantly, closed worlds (using the closed world assumption). [128, 57] The closed world assumption (CWA) assumes that everything that is not explicitly defined as true is false by default. On the contrary, the open world assumption (OWA) assumes that that what is not explicitly defined as true or false, is unknown, i.e. can be either true or false, but is not known. [116] Based on the following fact, that is defined as true: *Goran is from Pula.* – the answer to the following question: *Is Ozano from Pula?* – is false under the CWA, but is not known, and therefore neither true nor false, under the OWA.

Various languages have been used for defining ontologies, from Ontolingua to Knowledge Interchange Format (KIF), to RDF and OWL, with OWL2 defined as the latest recommended standard in ontology languages by W3C OWL Working Group [146]. OWL was thus chosen as the language for Lamrast−+ ontology implementation further detailed in this thesis, because of its role of a *de facto* standard in the domain of semantic modelling.

Conceptual models are extensively used in modern applications which demand increasingly dependable communication between human and artificial agents, or even amongst artificial agents without the access of a human agent. Many such modern examples are gathered by Karagiannis, Mayr and Mylopoulos [65].

Ontologies or the concept of semantic modelling are used in combination with MASs with increasing frequency, in many aspects related to MASs – whether for development of a metamodel, for simulations comprising many agents [9, 67], description of knowledge needed by agents in a system [101], enhanced understanding of a domain related to MASs and computer games, design of MASs using ontologies as the basis of the process [57, 129, 107, 114], semantic representation of agent plans and the planning domains [44], object annotations [10], or modelling smart city environments [17, 18, 74], to name a few.

Ontologies of the referenced examples are mostly used as models of knowledge that is available to agents of the observed system. Those ontologies that are designed to contain data further usable for modelling the observed system are concerned only with the basic features of such a system, e.g. description of agents and objects in the system. The use of ontology in this thesis is most similar to that presented in [17, 18], since both feature ontologies as the first step towards a defined metamodel for a specific domain – LSMASs in the case of this thesis. What is more, the ontology of this thesis contains concepts that can be used to describe organisational features of a group of agents, which is a purpose not seen in recent research. Finally, the ontology of this thesis is utilised as a medium for providing a clear and unambiguous definitions for the concepts of the metamodel and their extensions.

Other than using ontologies as a part of specific MASs, ontologies are used as knowledge maps for the various domains of computer games and organisation [118, 101, 145, 108, 98, 49]. The concept of organisational dynamics was not tackled yet though, as ontologies and models by default represent a real-world phenomena in a moment in time.

The ontology of this thesis can be further constrained to achieve specificity necessary for an efficient description of a gaming domain (e.g. to describe an MMORPG).

1.4.3 Models in the Domain of Multiagent Systems

Organisational models for MASs are used for showing organisational architecture of agent systems. Such models are explicated using modelling languages comprising specific symbols. Modelling language usually has two basic elements [5, 53, 59]: conceptualisation (a set of modelling concepts), and syntax (rules of using the conceptualisation elements). Therefore, a model [29] is an instance of syntactic conceptualisation of a given domain. A modelling language is defined using a metamodel, i.e. a model of a model.

When the combination of MASs and system modelling is concerned, there are two distinct concepts that have to be taken into account: multiagent systems (MASs), and agent-based modelling (ABM). While a MAS details how an agent is implemented and what are the implementation details, including their actions, features, and possibilities, an ABM is interested in observing agents' behaviour, interaction on a more social level, and how the involved agents act in the given environment. In other words, MASs are used more often in the context of development and integration of systems comprising a multitude of agents (both virtual and real everyday systems are of interest to this observation), while ABM is the concept often used alongside the concept of agents in the context of simulations and simulation models. Therefore, the ABM approach is commonplace in research on economics or social sciences, as a tool for conducting behavioural experiments that would be too expensive, technically complex, or morally complicated, to be performed on real subjects. Many of the arguments towards ABM in these respective fields are presented in [15, 24, 153].

A part of the ABM approach is research of various characteristics of agents and their allocation to roles. Roles in this context [130] group agents with appropriate characteristics, being their connection to different sets of tasks that are associated with roles. The modelling approach proposed by Sharpanskykh [130] is a form of ABM that fosters modelling motivation of an agent. Agent motivation is an interesting concept for research even in the context of belief-desire-intention (BDI) agents, and further social studies of MASs and use of MASs in social studies and related experiments.

A more recent study by Béhé et al. [9] proposed a metamodel based on an ontology for multiagent-based simulations. Using this metamodel, the simulation is split in description into two parts: a running ontology which encompasses all the entities related to or produced by the given simulation, and definition bases that define all the entities that can be encountered during the given simulation.

The first methodology that combines the benefits and good practice of both existing ontology and MASs design methodologies is presented in [57], under the name of Onto-

Agent Methodology. The second part of the methodology, i.e. the agent methodology, consists of the following five steps: 1) Classify agents according to their responsibilities; 2) Identify the need for an ontology to support agents' intelligence; 3) Define agents' collaboration; 4) Construct individual agents, 5) Protect the system by implementing security requirements. Other than reusing the best features of earlier methodologies, new characteristics are introduced as well. All of the methodology steps are described in [57].

A comprehensive, although not exhaustive, overview of modelling methods for MASs was published by Abbas, Shaheen and Amin [3]. The overview of those modelling methods for MASs, found in an organisational context, was presented in short in [92], where key concepts where each model's key concepts were extracted, as follows.

The first observed model, AGR model (agent/group/role, also known as Aalaadin) [41], features three key concepts – individual agents, group of agents, and agent roles. Agents are in the context of the AGR model considered as individuals capable of interacting and communicating with each other, independent of their levels of reactivity or intelligence. Since the model does not delve into implementation details of an agent, agents can simply enact roles or belong to a group of agents. Groups comprise many agents that share a common interest or a common feature. Thus, groups can be arranged to form organisational segments, whether in functional or structural sense.

TÆMS framework's [34] most prominent feature related to MASs is layered description of environments (a concept that is somewhat different from the standard environment concept in the context of MASs). Since the original intention of the framework was to model complex computational tasks, task analysis, environment modelling, and simulations, it provides concepts necessary for describing tasks and group tasks. Tasks and environments are characterised using three layers [34]: objective, subjective, and generative. Agents of this framework are not defined as usual either, being modelled as a locus of belief and action.

A model that builds on aforementioned Aalaadin model, MOISE+ (Model of Organisation for multI-agent SystEms (MOISE) [61], comprises concepts that are needed for structural, functional, and deontic modelling of organisation in the context of MASs. The focus of this model is on modelling roles and relations between them, rather than on modelling agents. Similar to a normative perspective, roles are considered as constraints that individual agents must follow when enacting a specific role. Roles are defined in a cause-and-effect style, with available roles being dependent on the role already played by an agent. Grouping is performed on a role basis, with agents enacting a specific role grouped in a specific group. MOISE+ features functional specification, wherein goals are structured in plans and grouped in missions. Plans are not necessarily linear, as they can be modelled using the available notation for sequential, parallel, or choice-based plans.

Although possibly deemed as an outsider in this group, a language for textual specification of electronic institutions, ISLANDER [40], is included here for its language parts that are used for defining performative structure, its scenes, and normative rules. Normative rules define action consequences, whereas possible actions an agent can perform are defined by roles as sets of constraints over individual agents. Some of the more important constraints are communication protocols, which are used in inter-agent communication.

OperA framework [35] is focused on describing a system at a conceptual level, comprising concepts whose main use is to define structure and global behaviour of a model. Agents of the system are modelled independently and separately, when their internal design is considered. Three components of the framework are considered: organisational model (models roles and interactions), social model (distributes individual agents along the defined organisational stucture), and interaction model.

AUML [141] is an agent-based extension of UML that incorporates swimlaned, class diagrams, sequence diagrams, and activity graphs. Swimlanes are to be used for role grouping purposes. Class diagram serves for the purpose of defining roles and their relationships. Finally, sequence diagrams are used for describing possible interaction withing the modelled system, amongst the defined roles.

The model which set out to be the most general one of all in this list, NOSHAPE MAS [1, 2], works with three levels of abstraction: universe, world, and organisation. It is important to note that this model knows about holarchy and hierarchy, wherefore individuals can be either individual agents or a group, depending on the perspective. A similar approach is used in the definition of an organisational unit in the context of Lamrast−+ metamodel.

MACODO [154, 155] is another particularly interesting model because its main intention is describing dynamic organisations. Agents are therefore modelled separately from their lifecycle, which is a technique that makes it easier to understand and model changes in a given system, or its environment, as well as their effect on the given system and its elements.

Overview of the above models is purposed as a short description and depiction of the most common concepts used in models for MASs – those for describing structure of a system of agents (e.g. groups of agents, and agents), interaction within the system (e.g. communication protocols), normative restrictions (e.g. norms as roles), and some functional features of an organisation (e.g. capabilities of agents), to name a few.

All the models mentioned in the overview above, except for NOSHAPE MAS, the most recent one, are used to deal with MASs, and not their large-scale counterparts. Several levels of abstraction, mentioned by NOSHAPE MAS help coping with large-scale systems, which is the intention of the Lamrast−+ metamodel as well.

Development of various aspects of MASs or LSMASs exists in the form of a large number of diverse models or platforms. Apart from those briefly described above, others are available as well. One such, jTRASTO (java Real-Time Agent Specification Toolkit) was published by Navarro, Julian and Botti [90]. As the name suggests, this framework provides developers with the opportunity to develop real-time MASs in accor dance with jART (java Agent for Real-Time) platform. The authors argue about the popularity of Java programming language for implementation of MASs, although Java at the time (year 2007) was not apt for implementation of real-time systems, such as the one proposed by jTRASTO, until Java extensions were developed that deal with garbage collection, dynamic load of classes, and general stability of the given system. However, byways do exist, wherefore the implementation of jTRASTO is possible and feasible. Afterwards, in year 2015, the jART platform was used by the authors to implement a system of real-time agreement agents [91].

Another approach to modelling MASs was published by Horling and Lesser [60], in the form of the Organisational Design Modelling Language, which renders models in two distinct forms: a template that contains explicit encoding of organisational decisions that must be made, and an organisational instance which is based on the defined template, and created by making specific choices for the defined decisions. The language contains a set of concepts, including node templates, parameters, *has-a* relations, constraints, variables, etc.

When recent publications are considered, it is worth noting a three-layer platform for large-scale game-playing multiagent systems on a high performance computing infrastructure developed in JAVA that focuses on large-scale machine learning experiments [68]. Another novel model featuring some organisational aspects, such as norms, roles, organisations, and interaction, in the context of designing holonic multiagent systems (HMASs) with added normative concepts in order to retain the idea of social control within such systems, is described by Missaoui et al. [83]. In the context of MASs built for the purpose of modelling and simulations of large-scale complex adaptive systems, Birdsey, Szabo and Falkner [13] introduce the specific concept of *semantic groups* to an already published earlier defined language, representing a group of agents that have a semantic relationship. Thus enriched language can be used to define applicable systems. Models developed to foster the concept of self-organisation (briefly mentioned in Section 1.4), such as the one described by Lhaksmana, Murakami and Ishida [73], emphasise the importance of roles, for self-organised systems, by default, do not have their goals and behaviour of agents defined beforehand. Furthermore, some recent studies [72] referenced in [3] introduced a temporally-dynamical organisational component to organisational models for LSMASs arguing the need for real-time models for LSMASs and promoting reorganisation in LSMASs.

Lamrast−+ metamodel is either a more generalised view on many of the just mentioned examples, or is complementary with them. While the three-layer platform for large-scale game-playing multiagent systems [68] is focused on grid infrastructure and faster or automated execution of experiments thereon, the research of Missaoui et al. [83]

concentrates on normative elements used to constrain a system of agents. Applicable to LSMASs due to the holonic point of view, which is similar to that of Lamrast−+ metamodel, the model is used for defining a set of norms that constrain behaviour of agents in a system of agents. Such norms can be described using modal operators for obligation, permission, and designating something as forbidden. Although Lamrast−+ recognises the benefits of using holonic approach to defining groups of agents, normative aspects of organisations are contained in role definitions, while additional normative elements can, at the moment, be stored in a knowledge artefact. A difference between organisational and non-organisational norms does exist though. Semantic groups specified in [13] can be considered as a specification of a compound organisation unit of Lamrast−+ metamodel, since they represent a group of agents organised using a specific criteria of organisation (semantic relationship, i.e. a set of relationships between entities). It is interesting to note here that semantic groups, since they depend on agent relationships, are susceptible to time, yet so are compound organisational units of Lamrast−+ metamodel, using the concept of organisational dynamics. Finally, the role model presented in [73] is about modelling roles only, therefore in a way similar to the mentioned normative model [83], featuring no concepts that can be used for modelling organisations, role actions, or strategic elements such as objectives – all of which are featured in Lamrast−+ metamodel.

Coutinho, Sichman and Boissier [29] evaluate organisational models for MASs using the following set of dimensions: organisational structure, organisational functions, organisational interaction, organisational norms, organisational evaluation, organisational evolution, organisational environment, organisational ontologies. Out of the eleven analysed organisational models, most of them feature concepts of organisational structure, yet only a smaller number of models comprise concepts of organisational interaction, organisational functions, and organisational norms. It is expected that the four mentioned basic dimensions are present in most of the analysed models, whereas the four additional dimensions are often missing. Indeed, only two models contain concepts of all the basic dimensions (e.g. OperA, MAS-ML).

A newer research done by Schatten [118] and Schatten, Ševa and Tomičić [122] presents a revised set of organisational modelling perspectives that are argued to aid more to building efficient LSMAS constrained by organisational features. The mentioned set contains the following seven perspectives: organisational structure (decision and information flows of an organisation), organisational culture (important intangible aspects of an organisation including knowledge, norms, reward systems, language and similar), strategy (long term objectives of an organisation, action plans for their realisation as well as tools on how to measure success), processes (activities and procedures of an organisation), individual agents (the most important asset of any organisation – individual agents actually performing the work), organisational dynamics (organisational changes including reorganisation of any of the mentioned components), as well as context and inter-organisational aspects (organisational behaviour towards its environment including strategic alliances, joint ventures, mergers, splits, spinouts, and similar).

Based on the above, Lamrast−+ metamodel represents a novelty insomuch that it:

- features concepts necessary for high-level organisational modelling of LSMASs, as opposed to modelling MASs only;
- respects the recursive nature of holonic point of view on the concept of organisational unit;
- allows model designers to model normative elements of a system using the concepts of roles and their actions, storing further norms in knowledge artefacts that are accessible to to either organisational concepts (i.e. roles), or individual concepts (i.e. individual organisational units);
- provides concepts that can be used for modelling simple and complex objectives, and their position in an organisation;
- can be used for defining various forms of agent groups (compound organisational units), independent of their used criteria of organising;
- provides the concepts for modelling a system of agents that can dynamically change the actions at their disposal based on the roles they enact.

Apart from the beneficial features of the Lamrast−+ metamodel, further benefits are apparent in the provided modelling tool, the most prominent one being the implementation template generator.

Chapter 2

Scientific Contribution

The following chapter is concerned with the scientific contribution of the research presented in this thesis, divided in the following manner [94]:

- a domain-specific **ontology** comprising organisational concepts applicable to the domain of large-scale multiagent systems (LSMASs);
- a **metamodel** that allows for modelling of various application domains of LSMASs, especially massively multi-player online role-playing games (MMORPGs), emphasising modelling of organisational dynamics.

Both of the above stated elements are presented in this thesis starting with a short introduction to the element at hand. A detailed account of the process of development of the respective element follows, extended with an elaborate description of the associated concepts. Both of the scientific contribution sections end with an example serving evaluation purposes.

2.1 Semantic Modelling

2.1.1 Ontology Engineering Methodology

A comprehensive overview of ontology engineering methodologies authored by Iqbal et al. [63] sorts and marks various ontology engineering methodologies based on eight of their attributes:

- type of development;
- support for collaborative construction;
- support for reusability;
- support for interoperability;
- degree of application dependency;
- life cycle recommendation;
- strategies for identifying concepts;
- details of methodology.

The set of 16 ontology engineering methodologies [63], evaluated against the above set of criteria, is shown here in Table 2.1. Type of development classifies each ontology engineering methodology into one of the following three categories: stage based model (useful when the developer has purpose and requirements clearly defined), evolving prototype model (used when evolution is favoured, as requirements are not clear from the beginning), and guidelines (provides the ontology engineer with useful tips, rules, and techniques for achieving better results, rather than presenting them with an overall development model). The second criteria aims to foster the Internet as a collaborative tool, and indicate whether a given ontology engineering methodology can be used for collaborative ontology construction, thus allowing geographically or otherwise varied team members to work on a single ontology at the same time. Support of reusability, as the third criteria, indicates if the given ontology engineering methodology supports reusability of concepts of existing ontologies. The fourth criteria indicates whether an ontology engineering methodology supports interoperability between systems. Application dependency criteria categorises each ontology engineering methodology as either dependent, semi-dependent, or independent of a specific application, when ontology development is considered. The sixth criteria indicates if a life cycle is proposed or not. Whether a strategy for identifying concepts is defined, is indicated by the seventh criteria, with three possible classes: bottom-up approach, top-down approach, or a middle-out approach. The final criteria classifies an ontology engineering methodology according to the details provided: insufficient detail, some detail, and sufficient detail.

METHONTOLOGY ontology engineering methodology [42] was chosen based on the review of ontology engineering methodologies published in [63]. The chosen methodology is based on a developing prototype, has reusability support, is not dependent on a specific application environment, is very well described, and has a clear ontology life cycle recommendation. The above features of METHONTOLOGY are clearly, and comparatively with some other ontology engineering methodologies, laid out in Table 2.1.

Being based on a developing prototype, the chosen ontology engineering methodology predicts that the final product i.e. an ontology, is not generated in an instant, but is a result of many iterations, and a process of refinement. Therefore, the overview of the METHONTOLOGY steps given in Fig. 2.1 is not a waterfall-like list of steps, but represents a collection of steps with their natural, but not exclusive or restrictive, sequence. One of the most valued advantages of such an approach, in the context of this specific research, is the refinement process foreseen by default in a way that the ontology is not finished until the author is satisfied with the result, possibly using feedback from some of the steps following any of the methodology steps (except, for example, the maintenance state). Moreover, it is stated in [63] that *evolving prototype may be the best choice when requirements are initially not clear and need refinement over time.*

As opposed to some of the ontology engineering methodologies on the list of [63], METHONTOLOGY allows the ontology engineer to use ontologies that already exist, thus avoiding building theirs from scratch. Such an approach is useful in this particular scenario, since the ontology of this research is based on work already published in various papers [126, 101, 98, 111, 104]. Furthermore, one of the basic features of an ontology is the principle of interconnectedness, i.e. most ontologies available online are meant to be reused by default [140, p. 7]. Therefore, a methodology that includes reusability support is highly appreciated for this research. Iqbal et al. [63] states that methodologies that support reusability help ontology engineers reduce the time and effort necessary to develop an ontology, leaving them with more time to spend on other issues, such as assuring ontology quality.

Even though the application scenarios for the ontology of this research is stated in the basis of this thesis, using an application independent ontology engineering methodology broadens the potential of the finished ontology, as it does not constrain the engineering process based solely on the application scenarios and specific application knowledge base. Furthermore, even though applicable scenarios are stated, the ontology should eventually be used in various ways and with numerous kinds of agents, systems, etc.

Table 2.1: Structured comparative overview of ontology engineering methodologies presented in [63]

Methodology	Type of development	Collaborative construction	Reusability support	Interoperability support	Degree of application dependency	Lifecycle recommendation	Strategies for identifying concepts	Methodology details
TOVE	stage based	no	yes	no	semi independent	no	middle out	some
Enterprise model approach	stage based	no	yes	no	independent	no	middle out	some
METHONTOLOGY	**evolving prototype**	**no**	**yes**	**no**	**application independent**	**yes**	**middle out**	**sufficient details**
KBSI IDEF5	evolving prototype	no	yes	no	independent	no	not clear	some
Ontolingua	modular development	yes	yes	yes	independent	no	not clear	some
Common KADS i KACTUS	modular development	no	yes	no	dependent	no	top down	insufficient
PLINIUS	guidelines	no	no	no	independent	no	bottom up	some
ONIONS	modular development guidelines	no	no	yes	semi independent	no	not clear	insufficient
Mikrokosmos	guidelines	no	no	no	dependent	no	rule based	some
MENELAS	guidelines	no	no	no	dependent	no	concept graphs	insufficient
SENSUS	does not mention any preference	yes	yes	yes	dependent	no	bottom up	some
Cyc methodology	evolving prototype	no	yes	no	independent	no	not clear	some
UPON	evolving prototype	no	yes	no	independent	yes	middle out	some
101 method	evolving prototype	no	yes	no	independent	no	developer's consent	some
On-To-Knowledge	evolving prototype	no	no	no	dependent	yes	middle out	some

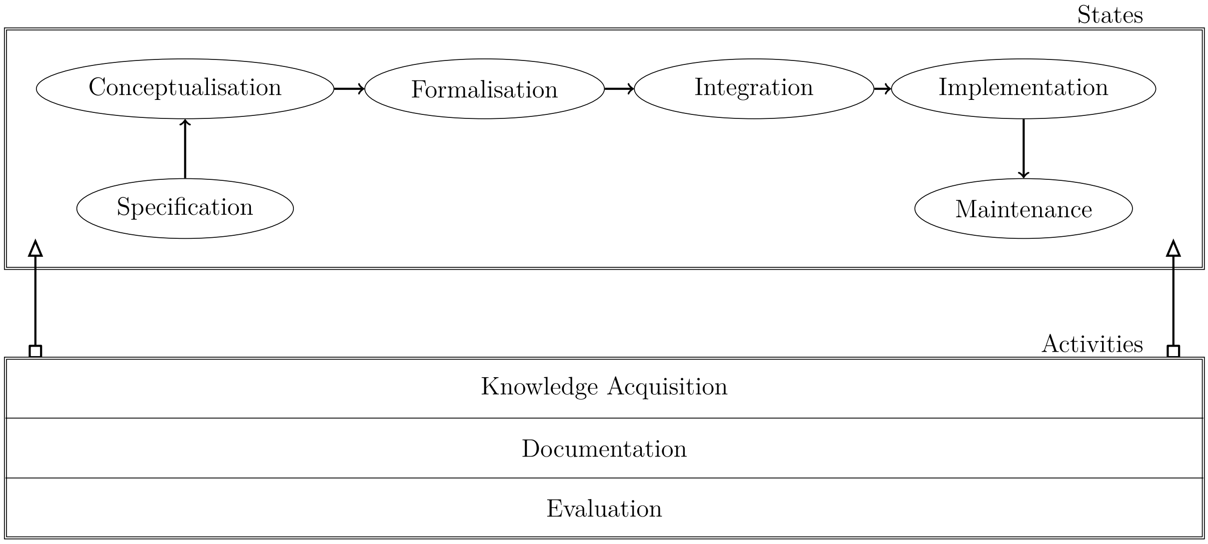


Figure 2.1: Basic steps of METHONTOLOGY ontology engineering methodology, adapted from [42]

Using a set of methods, just like initial use of a metamodel without former experience in using it, can be a tedious work, if the set of methods, or a metamodel, is not described clearly enough and in sufficient detail. METHONTOLOGY is given a thorough explanation and description in the initial paper of its original authors [42], and is even given a slight modification in a later publication [75]. Having a summary of its steps, states, and various included instructions at hand can prove to be a useful guiding element to how a methodology is designed to be used.

> "Methodologies classified to have sufficient details cover the employed techniques with reasonable level of details, allowing the reader to clearly understand the technique and its application in the ontology development process."
>
> — Iqbal et al. [63]

In addition to being clearly described, the authors of METHONTOLOGY published a set of identified states or stages in a life cycle of an ontology developed using their methodology. In the context of developing an ontology, an ontology life cycle is defined by Iqbal et al. [63] as a *set of stages through which the ontology moves during its life.* Using an ontology engineering methodology that proposes stages in a life cycle of an ontology provides the ontology engineer with a sense of security, since the guidelines do not stop immediately after the ontology is created.

2.1.1.1 Activity One: Specification

> "The goal of the specification phase is to produce either an informal, semi-formal or formal ontology specification document written in natural language, using a set of intermediate representations or using competency questions, respectively."
>
> — Fernández-López, Gómez-Pérez and Juristo [42]

As stated above, the goal of the specification phase or state (as referred to in Fig. 2.1) is to define an ontology specification document. According to the guidelines about the proposed information included, the Lamrast−+ ontology is specified as follows.

The Lamrast−+ ontology comprises various selected concepts from the organisational modelling domain applicable to LSMASs. The main purpose of the ontology, strictly speaking in the confines of this thesis, is to serve as a basis for the definition and development of the Lamrast−+ metamodel for organisational modelling of LSMASs. From a broader perspective, the purpose of the Lamrast−+ ontology is to collect and structure concepts of human organisation modelling domain that are applicable to the domain of LSMASs so as to facilitate organisational modelling of LSMASs, since, as stated earlier, modern and upcoming instances of LSMASs in various application domains benefit from organisational features (e.g. an organisational structure), and self-organisation, especially in the context of achieving a system-wide common goal and utilising multitude of agents' abilities.

The Lamrast−+ ontology is used as an initial stage of the Lamrast−+ metamodel, yet its use is not limited to such a scenario, but can be broadened to description of various application domains of LSMASs for many purposes, such as, but not limited to, simulations, analyses, knowledge repositories, etc. As such, the ontology is, certainly, intended to be used by developers (even game developers), knowledge engineers, and designers of LSMASs.

Therefore, the ontology must comprise, at least, the following elements:

- a list of elements used in organisation modelling, e.g. OrganisationalUnit, Organisation, OrganisationalArchitecture, OrganisationalDesign, Goals, OrganisationalStructure, etc.;
- a list of particular examples to depict use case scenarios of this particular ontology;
- various properties necessary for clear modelling of a specific domain, e.g. isAPartOf, contains, isDefinedBy, isOrganisationalStructureOf, etc.

While developing the Lamrast−+ ontology, a middle-out approach is used, as opposed to the classic bottom-up or top-down approaches, since it makes it possible to identify the primary concepts early on, moving to specialisation and generalisation afterwards if necessary, which results in less re-work needed and increased stability of the whole process. [42]

It is necessary to mention here that the Lamrast−+ ontology is not concerned with detailed modelling of individual agents, or their behaviour and complexity. Organisational units are basic building blocks of an organisation, and their realisation is of no concern when developing the ontology. In line with the mentioned, the Lamrast−+ ontology provides a recursive definition of an organisation, and by extension an organisational unit.

Such an approach is elaborated in [118], where organisational units are defined clearly as individuals on the lowest observable level, that build an organisation supported by several features of an organisation. These organisations can be organised into higher-level organisations, whereat they can be considered organisational units. An easy illustrative example of the above observation follows. In a smart city domain, an apartment in a residential building can be occupied by smart appliances alongside human tenants. These appliances (organisational units) can be grouped into an organisation on the apartment level. Three other such apartments are located on the same floor of the observed building. All the apartments of the same floor can be grouped into a floor level organisation (the apartments are therefore now considered organisational units). The same approach can be applied to other interesting levels of the observed workspace, e.g. floors organised into the whole building. A very similar example is presented by Tomičić, Okreša Đurić and Schatten [136].

At the end of this activity, the above section represents an ontology requirements specification document, in the domain of organisational modelling of LSMASs.

2.1.1.2 Activity Two: Knowledge Acquisition

> "[...] knowledge acquisition is an independent activity in the ontology development process. However, it is coincident with other activities."
>
> — Fernández-López, Gómez-Pérez and Juristo [42]

The paper detailing activities of METHONTOLOGY and the related methods [42] presents the reader with a number of techniques its authors used while developing a chemical ontology. The following techniques are mentioned:

- interview with experts (both structured and unstructured),
- text analysis (both formal and informal).

Additional techniques, including brainstorming and knowledge acquisition tools, are recommended by the authors.

As the knowledge acquisition activity is not a time-constrained activity, i.e. it can be performed throughout the most of the ontology engineering process, it already started during the Specification activity, thus not being strictly second, but an underlying activity of the whole process.

Knowledge acquisition techniques for the Lamrast−+ ontology engineering process are quite simple in their most basic form, but get rather complicated in the context of content analysis, comparison, and selection.

The main body of knowledge that is used in development of the Lamrast−+ ontology was created during the cro. Organizacijsko oblikovanje višegentnih sustava u Internetu Stvari - eng. Organizational Design of Multi-Agent Systems in the Internet of Things

(OOVASIS) project, and is hereafter referred to as OOVASIS ontology. The OOVASIS ontology [126, 118] consists of the concepts that were identified as useful for organisational modelling of LSMASs, by direct transfer of such concepts from the set of concepts used in describing human organisations.

The OOVASIS ontology and the underlying work is described in some of the published papers, mainly [126, 118], and the ontology itself, along with some of the more elaborate descriptions available at the project wiki website. In order to access and assess relevancy of all the available content, all the available sources have to be analysed. The key piece of information about the OOVASIS ontology is that it comprises all the concepts that were identified in the domain of human organisations and directly applied to the domain of LSMASs. This approach is good in generally observing organisational modelling of LSMASs, but the ontology developed for the purposes of this research and the emerging results can be further constrained, or reduced in the number of available and important concepts. Discussion on OOVASIS is continued in Section 2.1.1.4.

The second very important knowledge resource of concepts concerning (organisational) modelling of LSMASs comes in form of the Multi-Agent Model For intelligent virtual environments (MAM5) metamodel and ontology. MAM5 is based on the idea of LSMASs, but with a more emphasised organisational approach, and providing a development environment of sorts when used in tandem with Jason Cartago implemented intelligent virtual environment (JaCalIVE) framework, which *provides a method to develop a kind of intelligent virtual environments (IVEs) along with a supporting platform to execute them* [112], and is based on the MAM5 metamodel. Considering how appropriate MAM5 is for the topic of this document, it is considered noteworthy in the context of knowledge acquisition activity towards the Lamrast−+ ontology.

Further knowledge acquisition tasks are performed related to the activity of integration, Section 2.1.1.4, since already existent models for multiagent systems (MASs) and LSMASs have to be found, identified, and analysed.

Certainly, the knowledge acquisition activity is not a one-shot activity, and is being performed all through the research, thus allowing for further improvement of the already done work. Probably owing to such a nature of this activity, knowledge acquisition is primarily performed using the available text analysis techniques, with some basic unstructured expert interviews where applicable.

2.1.1.3 Activity Three: Conceptualisation

> "[...] you will structure the domain knowledge in a conceptual model that describes the problem and its solution in terms of the domain vocabulary identified in the ontology specification activity."
>
> — Fernández-López, Gómez-Pérez and Juristo [42]

It is noted in [42] that the conceptualisation phase of METHONTOLOGY is about producing a set of well-defined deliverables that make the act of ascertaining whether the final ontology is needed at all, useful, and usable for the given application domain. Furthermore, if these deliverables are done well, ontology comparison can be done based on them.

The initial glossary of terms (GT) includes all the relevant concepts, instances, verbs, and properties of the given domain. More specifically, GT by definition *identifies and gathers all the useful and potentially usable domain knowledge and its meanings.* [42] Such a GT is based on the specification document, i.e. the result of the specification activity described in Section 2.1.1.1.

Further specifics of the conceptualisation activity are defined using some of the following [42, 49]:

- data dictionary (DD) – used to describe all the gathered, useful, and potentially usable domain concepts, their meanings, attributes, instances, etc.;
- tables of instance attributes – provides information about attributes or their values at the instance level;
- tables of class attributes – similar to the above, but at the concept level;
- tables of constants – specifying features that never change;
- tables of instances – defines relevant instances;
- attributes classification tree – graphical representation of *attributes and constants related in the inference sequence of the root attributes, as well as the sequence of formulas or rules to be executed to infer such attributes.* [42]

Whilst the above mechanisms (shown graphically in Fig. 2.2) are used for concepts, identified verbs from the GT, that represent actions in the given domain, are handled and described using the following:

- Verbs Dictionary – declarative expression of the meaning of relevant verbs;
- tables of conditions – specifying the pre- and post-conditions of relevant actions.

In case there are identified formulas or rules, a table of formulas, and a table of rules, are defined, so as to gather the available knowledge about formulas and rules.

data dictionary Concepts identified in this step of METHONTOLOGY, that are a part of the data dictionary, are presented in Appendix A.1. Many of the identified concepts have their descriptions and definitions stated (e.g. concept A.20 Norm), and concepts' synonyms and acronyms are noted where applicable (e.g. concept A.3 Agent). Only those concepts that were afterwards classified as needed for the Lamrast−+ metamodel

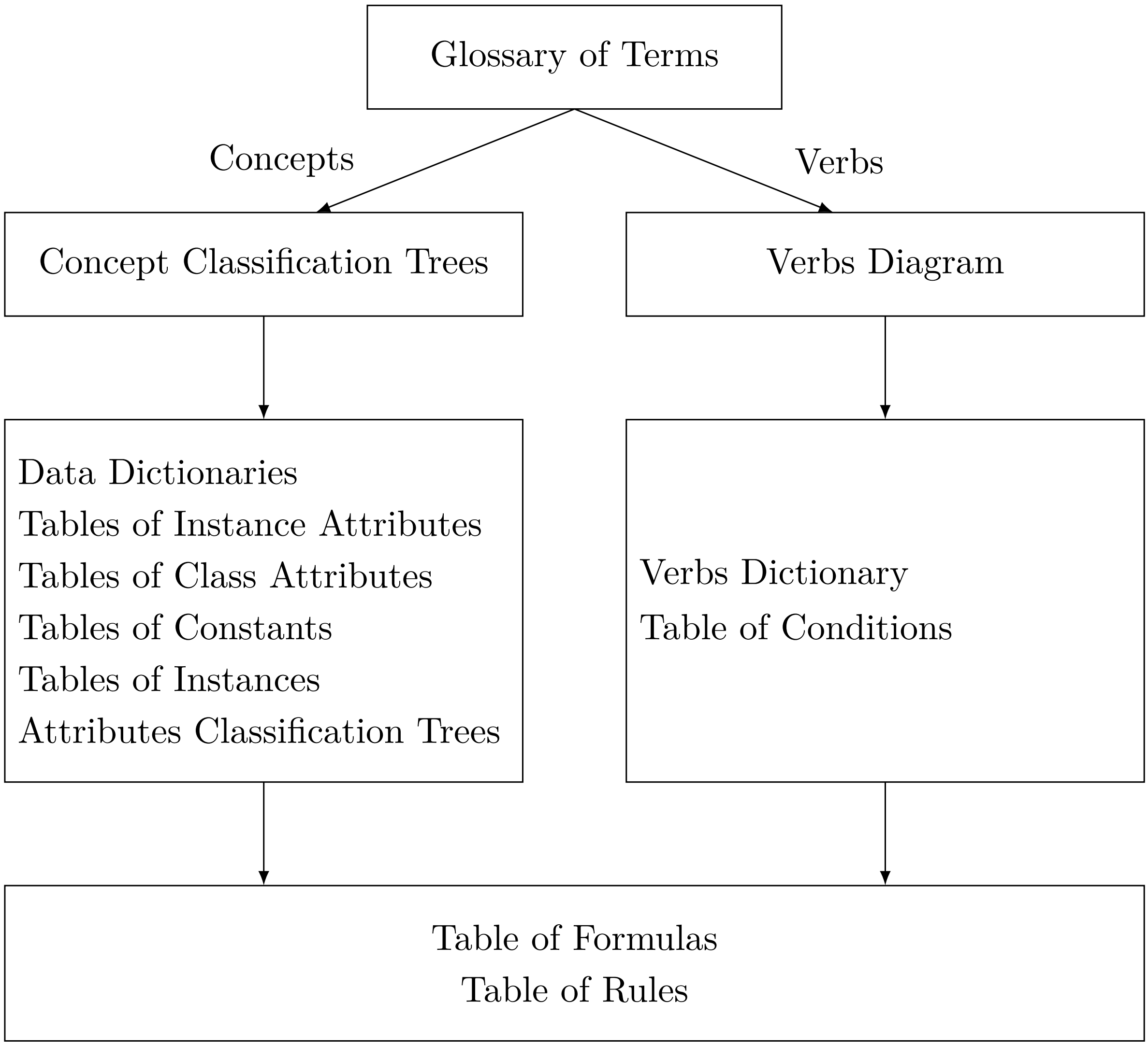


Figure 2.2: Intermediate Representations in the conceptualisation phase, adapted from [42]

development are detailed using class attributes and instances in the context of The Mana World (TMW), the open-source 2D MMORPG game used during the Large-Scale Multi-Agent Modelling of Massively On-Line Role-Playing Games (ModelMMORPG) project. The following is a brief description of a select number of concepts presented in DD, in the context of MMORPGs and LSMASs.

The identified concepts, as expected, mostly revolve around the idea of an organisation (A.24) as a group of agents structured according to a set of criteria (A.5). The main reason why such a group is constituted in the first place is the claim that an organised group of individuals is more successful than an unorganised group of individuals, mainly because organisation overcomes various hindrances of individual agents (A.3). Some of the mentioned hindrances are spatio-temporal uniqueness of an individual (one agent can at exactly one point in time be at exactly one point in space), skill constraint (no agent can be a *Jack of all trades*), and more [3, 4, 62, 46]. In the context of MMORPGs, the most common criteria of organising are certainly goals, i.e. common ground of a couple of player agents who are working each on their own towards fulfilling a certain goal, but realise that together the given goal can be achieved in an easier fashion, usually more profitable for all included. Such a grouping behaviour is observable in the context of creating short-termed groups usually called *parties*, while coalitions based on a more strategic outlook, thus longer lasting, are called *guilds*.

Organisations as systems comprising individual agents are encompassed by the concept of a Workspace (A.45) which comes from the MAM5 ontology. A workspace includes all the concepts even remotely relevant to an organisation. Physical section of the said concepts belongs to the concepts of IVE workspace (A.16). All the concepts that can influence the given organisation, but are not strictly a part of it, belong to the environment concept (A.28). The mentioned IVE workspace is one of the subconcepts of an IVE (A.14) – a virtual environment that simulates the real world, and is populated by autonomous intelligent entities. Various organisation of elements of an IVE causes organisations to change, in any of their key aspects, which leads to organisational change (A.26).

Individual agents are the basic unit of an organisation. These agents work using their actions (A.2) towards fulfilling high-level objectives (A.22) that are integral parts of a strategy of an organisation (A.41). Objectives can be cut down into goals (A.8), or quests (A.36) and tasks (A.43). A set of successive actions following the antecedent and precedent relations, i.e. such a set that has a positive outcome in the context of agent's aims, is a plan (A.34). A plan can be made true following a series of actions, i.e. a process (A.35).

Every action in a system in the context of this document pertains to a specific role (A.37) that can be played by an organisational unit. An organisational unit can, following the formal definition that uses a form of recursion (laid out in [118]), represent an individual agent, or a group of agents forming an organisation. Since a whole organisation, i.e. an organisational unit, can ultimately enact a specific role (like a wizard, or a builder), a role is observed as quite an abstract concept, employing the idea of norms (A.20). A norm is an informal rule that is socially enforced, and is a constituent part of a normative system (A.21). Although a general description of a normative system is given in the respective Appendix A.1, it is useful to state here that a normative system in the context of MAS is a blend of both normative systems and MASs. A normative multiagent system is therefore a set of agents that are governed by specific norms (i.e. their interaction is governed by norms) which can be obeyed, but can be deviated from as well. Thus, it is apt to state that agents can choose whether to follow the given norms, and that the system can decide upon the extent to which agents can modify the initially set norms. One could state that the instrument affecting the stated features is organisational culture (A.27).

The DD is subject to knowledge verification, so as to assure that there are no contradictory pieces of knowledge inside the DD, and that the contents are consistent throughout the DD. Aims of this process are enumerated in [49], and are immediately followed here by comments pertaining this particular DD:

- *To guarantee the completeness of the knowledge attached to each concept. That is, the concept description is concise and all the relevant instance attributes, class attributes and instances have been identified.*

  Concept descriptions and definitions are concise and clearly represent the aimed-at concepts. Attributes and instances are included only where their inclusion is beneficial towards fulfilling the research goal of this research, i.e. when concepts that are a part of the Lamrast−+ metamodel are considered.

- *To determine the granularity or level of detail of the concepts covered by the ontology.*

  From the defined descriptions and definitions, it is clear on what level of granularity a certain concept is being used, i.e. what level of detail it describes and can be used for.

- *Consistency of the instance attributes and class attributes. That is, they make sense for the concept.*

  All the concepts in DD have consistently named attributes that are divided amongst classes and instances in a consistent manner.

- *Concept names and descriptions. To assure absence of redundancies and to keep concision.*

  Concise concept names, descriptions, and definitions help in easy comprehension of the concepts included in DD. There are no overlapping concepts defined in the DD, yet when concepts are defined as almost identical, that is clearly stated, and their differences are emphasised.

Classification Classification [105, 99, 100] is a process that helps agents (artificial and real alike) observe and perceive their environment and structure their knowledge about it. It is used as a sort of a catalyst that fosters information communication, thus reducing the necessary amount of information agents have to remember, communicate and process. The extent to which the aforementioned is achieved depends on the number of properties of a concept [105, p. 39]. Therefore, data dictionary provides an overview of class and instance properties. Moreover, classification provides cognitive economy since it allows the agent to *structure knowledge about objects into two levels: concept and instance.* [105, p. 39]

Concept has properties common to all the instances of the said concept, while *at the instance level, we find only the concept of which the object is an instance.* [105, p. 39] Properties are classified as defining or non-defining [105, p. 38], where defining properties are *the necessary and sufficient properties for an object to be considered an instance of the concept*, and non-defining properties are described as redundant. The operation of classification is therefore simply mostly checking that all the defining properties of a concept are included in the set of all the properties of the given object, i.e. an instance must have all the properties of the given concept, optionally enriched with additional properties. Such a set of defining properties of a given concept is called the intension of the given concept.

Using the idea of classification, the concepts in DD were analysed, and their respective class and individual properties were defined. Such properties are listed in Appendix A.1, at their designated places. However, not all the concepts described within the DD have their respective class and individual properties stated. Only those concepts that are selected to be included in the finalised metamodel have their respective properties detailed.

Individuals of some of the concepts present in DD are named in Appendix A.1 under the Instance/s part of an applicable concept. The individuals stated there are mostly from the domain of The Mana World or MMORPGs. These individuals have or can have the individual properties stated in the Attributes part of select concepts. Only a small number of concepts are detailed using the possible instance attributes, namely those that are featured in the metamodel concepts. Most of the concepts that have no individuals in Appendix A.1 are described as abstract classes, having classes as individuals, or simply can have individuals that are not of interest for this research.

Most of the individuals in Appendix A.1 are references to work published primarily in [126], yet some reference the mentioned computer game of MMORPG genre, The Mana World. Such individuals as non-player characters (NPCs) Archmage and Sorfina, classified as Inhabitant Agents in DD (A.13), The Quest for the Dragon Egg, classified as a quest (DD A.36), and Wizard or Warrior as individuals of class Role (DD A.37) are prime examples of individuals from the MMORPGs domain. Individuals pertaining to another take on IVEs, MAM5, are present in extension sets of some of the classes, most interesting being the mentioned inhabitant agent (which is clearly well blended in the MMORPGs domain), along with IVE Law (DD A.15) represented with the individual *When a character is located on a map with at least 75% of tiles of type Frozen, they are more susceptible to Damage of type Ice.*, and intelligent virtual environment itself (DD A.14) with its individual that represents the whole modified version of the mentioned TMW computer game.

Tables of Instance Attributes It is stated in [49] that an individual instance attribute table is to be defined for every attribute included in the DD. Only the most interesting concepts of the DD have their instance attributes defined, as follows.

Out of all the parts of an instance attribute table prescribed in [49], some of them are omitted in the following tables, since their inclusion is not regarded as beneficial for the purpose of the process of engineering the ontology of this research.

Since it is accustomed to talk about properties rather than attributes in the context of an ontology, when one is discussing a Web Ontology Language (OWL) 2 ontology, instance attributes tables are referred to as instance property tables in this document. Considering that most of the properties used in Lamrast−+ ontology, and all the properties in tables of Appendix A.2 , this document works with properties – data properties that connect individuals with literals, and object properties that connect pairs of individuals [146]. The term *instance attributes* references attributes, i.e. properties, that are applicable on the level of individuals. Opposed to this, class attributes are *relevant properties of the concept that describe the concept itself.* [49] All of the properties in Appendix A.2 are classified as belonging to the original concept of *individual attributes* since they are defined at the level of individuals, even though a concept, i.e. a class, is defined using those properties, but in the context of individuals which it contains, i.e. particular individuals that are to be reasoned to be individuals of that particular class. Furthermore, the concepts of Appendix A.2 are, in fact, object properties.

The properties described in Appendix A.2 are those that are a part of the ontology and are deemed useful for the metamodel, or are interesting since they belong to useful concepts of the DD.

Properties are described in Appendix A.2 using a subset of properties proposed by [49]. The property *isAchievedBy* that is used by individuals of class *Objective* is described because it creates a link between specific actions (DD A.2) and objectives (DD A.22). As stated before, any given action can achieve exactly one objective, even though an objective can be achieved by a single action out of a set of them (denoted by Cardinality attribute of the property). For example, in the domain of MMORPGs, an objective of killing a non-player character, a mob, can be achieved either by attacking the target with whatever weapons available, or by using other means that are permitted in a given situation (e.g. throwing the target off a cliff). Situations as the one described are not often applicable, and mostly happen during a regular attack.

Several other properties described in Appendix A.2 are interconnected. An organisational unit that represents an organisation (consists of a number of lower-level organisational units and organisational features) by its definition defines some roles (property *definesRoles*, IA A.49). Based on the defined roles, and roles that are already available in the given organisational unit, a set of roles is available to be played by lower-level units of that particular organisational unit (property *hasRole*, IA A.52). All the roles that exist in an organisational unit, are, presumably, playable by lower-level organisational units, i.e. every role in an organisation can be played by at least one organisational unit of that organisation. Roles that are played by a given organisational unit are designated using the *playsRole* property (IA A.53. Furthermore, actions that are available to the organisational unit playing a given role can be used to achieve (inverse property to *achievedBy*, IA A.46) certain objectives.

It should be discussed here that roles are by default playable by at least one organisational unit in an organisation, yet do not necessarily have to be played at every moment in time. One should observe a situation in which the role of a Wizard can not be played by any of the organisational units of an organisation if none of the units possesses the necessary skills. It should be decided then whether the role should be disbanded, and therefore not defined by the organisation anymore, or it should be allowed to exist even though no organisational unit exists in the organisation that can play the given role.

Concept Classification Tree Concept classification tree is used to organise recognised domain concepts , most of which are described and defined in the DD. The classification tree represents visual take on the taxonomy and relations of the selected concepts. The concept classification tree in Fig. 2.3 represents a selected set of concepts, since using the whole ontology, using all the defined concepts, would further decrease legibility.

2.1.1.4 Activity Four: Integration

> "Ontologies are built to be reused. [...] So, you should reuse existing ontologies."
> — Fernández-López, Gómez-Pérez and Juristo [42]

The activity four of METHONTOLOGY strives to fulfil one of the main goals of the concept of ontology – re-usability and a big network spanning many ontologies. In order to achieve the set goal, domain compatible existing ontologies have to be analysed and marked according to the level of their fitting into the concept of the ontology being developed. Lamrast−+ ontology concerns the domain of organisation and LSMAS. Some

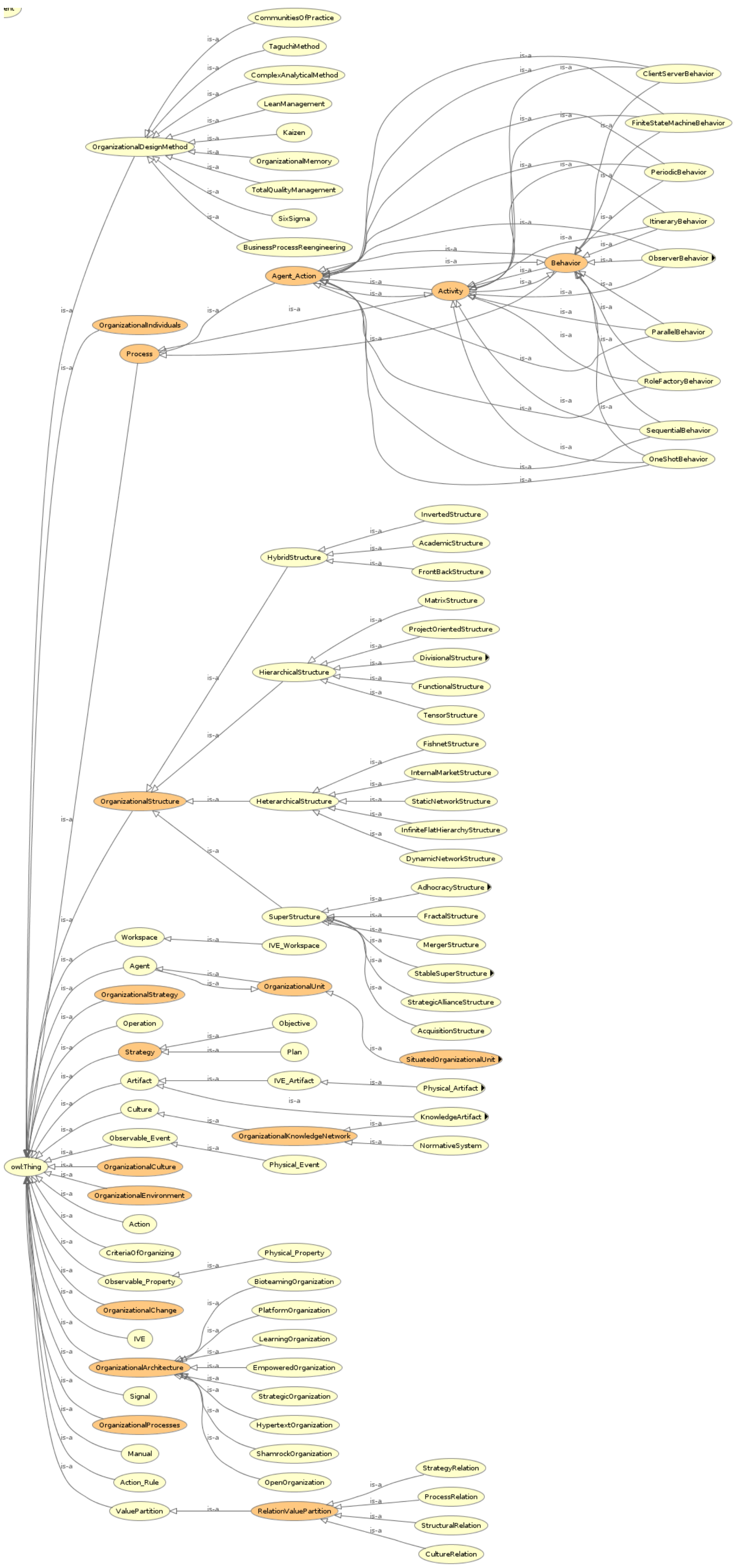


Figure 2.3: Concept classification tree

interesting existing ontologies were found to be connected to the said domain, yet three specific ontologies were decided to be most interesting and useful: OOVASIS, MAM5, and World Wide Web Consortium (W3C) *The Organization Ontology* [145].

OOVASIS ontology is deemed interesting because it represents a modern take on the problem of organisational modelling of LSMASs. MAM5 deals with IVEs, therefore posing as a good candidate as one of the most interesting existing ontologies, in the context of this research. The following is the account of the two most prominently included ontologies, in somewhat more detail.

OOVASIS ontology is a result of OOVASIS research project. The ontology comprises concepts applicable to the LSMASs domain, pertaining to the idea of organisational modelling of such systems, i.e. referencing various features used for describing human organisations. The research resulting in the OOVASIS ontology was conducted as a thorough study of publications in the domain of organisation theory, organisation architecture, and organisation design [126]. Customised tools were used to conduct the said research, detailed in [126, 118].

The original OOVASIS ontology was further developed and enriched during the course of ModelMMORPG project, a part of which is this thesis. Therefore, further in this thesis, the used ontology is the one that is the result of the ModelMMORPG project, but it is referenced here with the older prefix of `OOVASIS` for legacy reasons. This modified ontology [98, 101] is available at the project's web site[1].

One of the most prominent contributions of this research is the ontology featuring identified organisational concepts, from human organisations, applicable to the domain of LSMASs. Structure of the core concepts of the OOVASIS ontology is shown in Fig. 2.4, showing only concepts relevant to this document. Thus the research represents a theoretical groundwork for modern LSMASs using organisation features in LSMASs categorised into seven perspectives of organisational modelling primed for the future of LSMASs: organisational structure, organisational culture, strategy, processes, individual agents, organisational dynamics, as well as context and inter-organisational aspects. Another key feature of this document found its base in [118], that of recursive modelling of various organisational concepts, similar to the idea of holons and holarchy [113, 56, 83].

It is clear, considering the above, that the ontology is not concerned with individual agents, rather with organisational features of groups of agents or individual agents. Therefore, it can be used to describe mutual relations of agents in an organisational context and organisational features of an organisation formed by agents. The ontology still comprises concepts of human origin, i.e. pertaining to human organisations, an idea that is clearly transferred to Lamrast−+ ontology, even though it represents an open area for improvement, since not all of the identified concepts have to be present when artificial agents are taken into account.

[1]For more information, visit `http://ai.foi.hr/modelmmorpg.php`

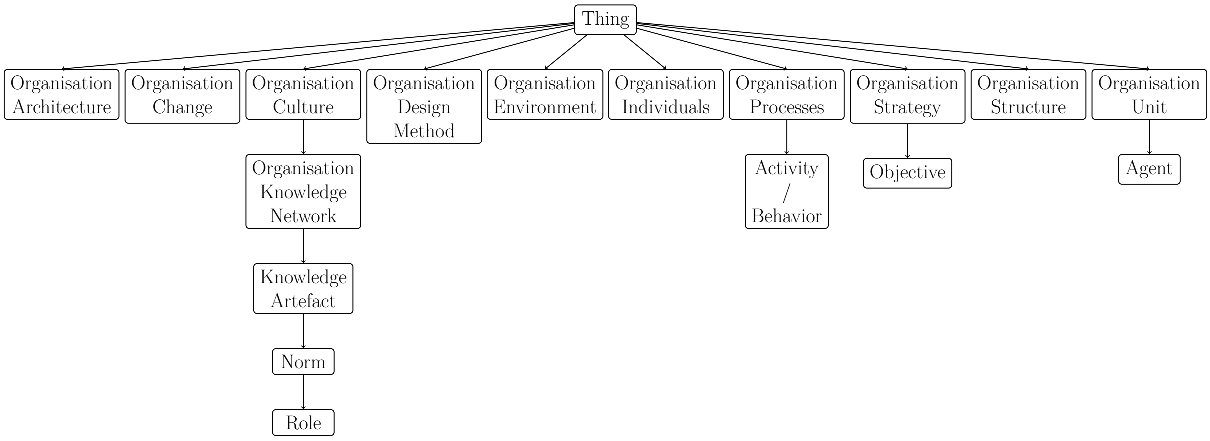


Figure 2.4: Visualised structure of core OOVASIS concepts. [104]

MAM5 is a model working with IVEs, featuring an ontology in its background that comprises concepts necessary for essential modelling of IVEs. An IVE is an abstract of a MAS, defined as a virtual environment simulating a physical world, inhabited by both human and artificial autonomous intelligent agents [8]. An overview of interesting concepts of MAM5 is given in Fig. 2.5, where concepts are shown that allow for modelling physical and non-physical elements of a IVE. Physical entities can be situated and represented physically in a physical world, as opposed to non-physical which cannot. Human-immersed agent is an interesting addition (in contrast to OOVASIS for example), emphasising that an IVE can include both artificial and human agents, or direct representations of human agents in a virtual system. MAM5 follows the Agent & Artefact (A&A) metamodel [106], thus allowing for representation of agents, artefacts, and workspaces. The artefact concept can be used to model various kinds of entities not classifiable as agents or workspaces (containers of system-wide elements).

Therefore, MAM5 ontology is a basis for the model that can be used for modelling virtual environments, on a declarative level. A limited palette of inter-system entity relations are available, yet more advanced concepts that would make modelling systems from the LSMASs domain possible are lacking. Such a state is possibly supported by a probable conclusion based on analysis of MAM5, stating that the said model is developed primarily to be used in the context of MASs, but not LSMASs. The former is motivated by observations of the importance of organisational features in LSMASs, discussed earlier in this document.

It should be mentioned here that the Lamrast−+ ontology is heavily based on the mentioned two ontologies, since both are applicable to the concepts of MASs, yet each covers a specific context of the mentioned domain. MAM5 is to be used to define systems comprising artificial and human agents, as well as artefacts, thus describing an IVE in terms of possible actions that agents can perform in order to affect their environment in a

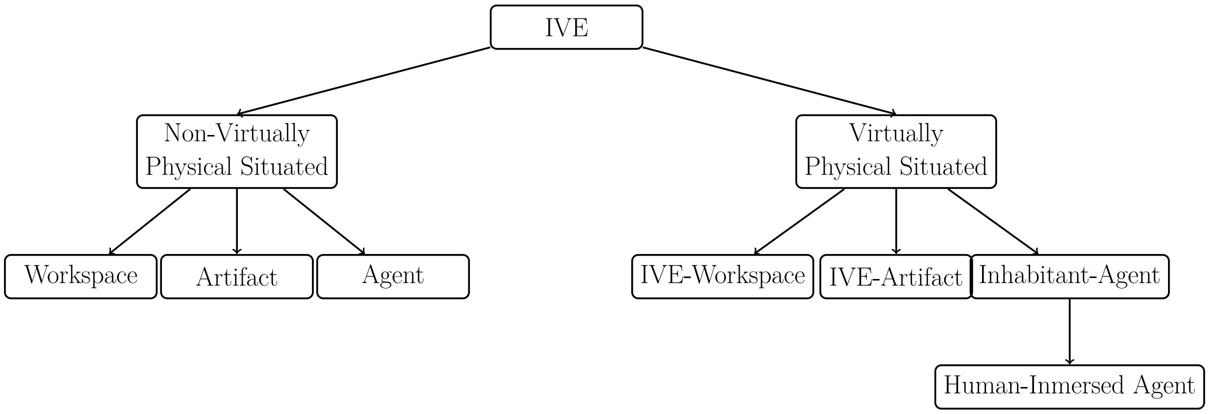


Figure 2.5: Visualised structure of core MAM5 concepts. [104]

predefined way. On the other hand, the OOVASIS ontology is defined in order to facilitate modelling of LSMASs, especially in the context of organisational features of such agents. Thus the emphasis is on the concepts necessary for expressing various organisational features of a system of agents. Furthermore, normative elements are included, fostering definitions of normative systems. Clearly, the two chosen ontologies are related by their common interest and primary domain, and their combination is seen as a useful addition to both of the ontologies. Further discussion on their individual benefits for the domain of LSMASs, as well as the benefit of their combination, is discussed further in this thesis and in [104].

The selected set of important concepts of Lamrast−+ ontology with noted original ontology names where applicable, i.e. where the given concepts were reused from another ontology, is available in Fig. 2.6. Every concept name is prefixed by the appropriate namespace, i.e. name of the ontology it comes from. `JaCalIVE` namespace denotes MAM5 ontology, `OOVASIS` the corresponding extended ontology, and `mambo5` is the namespace of an ontology that is a part of a collaborative research performed during the ModelMMORPG project, and was submitted for publication. It should be noted here that the Lamrast−+ ontology is a slight addition [98, 101] to already published research [8, 111, 126, 118], since it is focused on defining concepts applicable to the domain of LSMASs, thus being heavily dependent on concepts related to the domain of MASs.

Since the OOVASIS ontology is built with the LSMASs domain in mind, but with no real application or implementation examples, it presents a strong foundation for describing LSMAS applications in organisational context detailing their various organisational features. The lack of implementation details clearly keeps it in the theoretical domain. MAM5 ontology, on the other hand, is directed towards implementation by using the MAM5 model and the JaCalIVE framework in order to create a working example description of an IVE, which fundamentally models an LSMAS application domain example. The

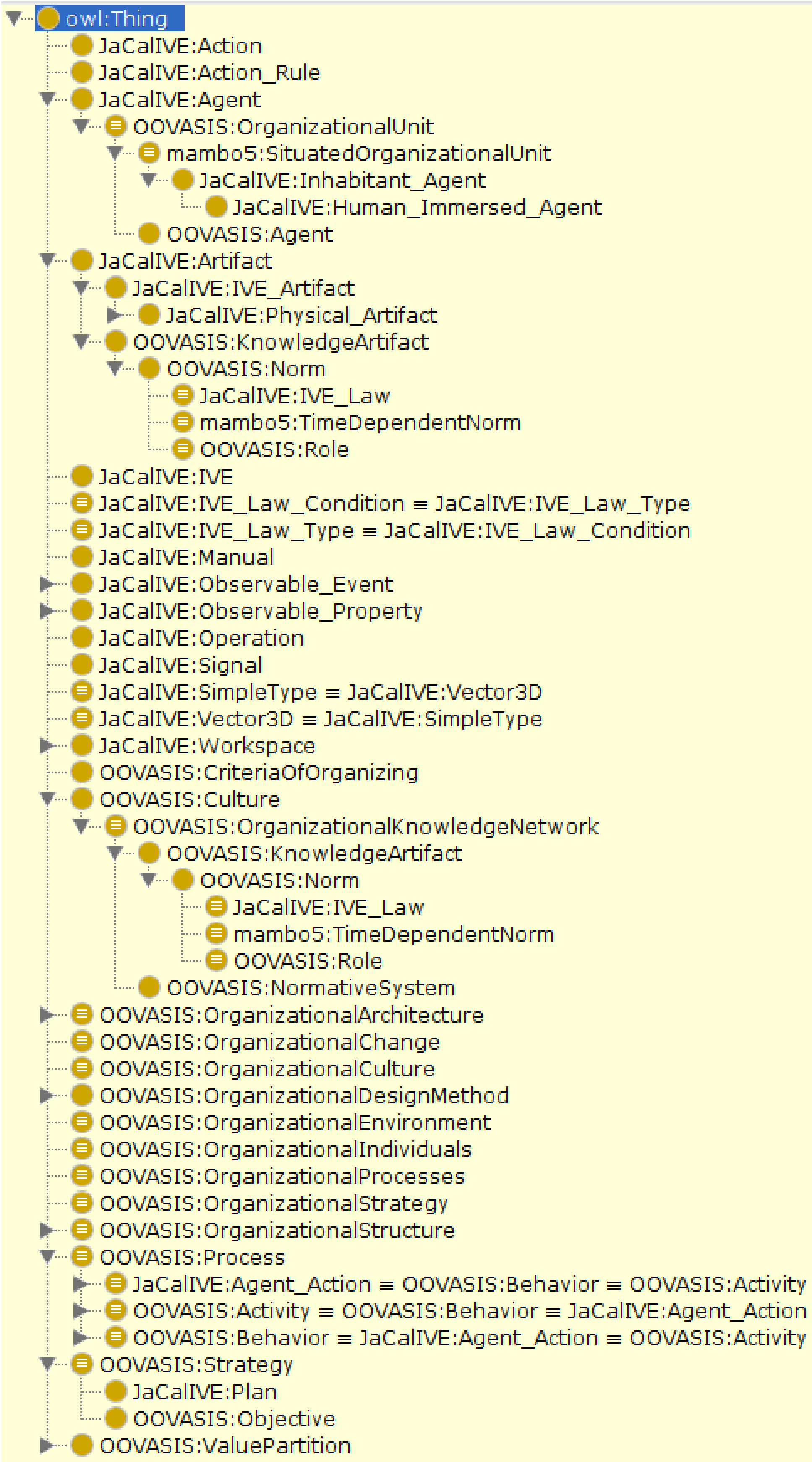


Figure 2.6: Lamrast−+ ontology class hierarchy as seen in Protégé

ontology presented in this thesis is utilised as a starting point towards defining a practically usable metamodel for organisational modelling of LSMASs, thus representing an upgrade on both of the referenced ontologies.

On both sides, from the perspective of MAM5, and that of OOVASIS, improvement opportunities are detected, in the form of further enrichment of the ontology, and further improvement of applicability or implementation, respectively. The benefits of introducing organisational features to MAM5 are identified in the opportunity to expand MAM5 to LSMASs, and modelling and implementation of more complex applications that would benefit from the introduced organisational concepts, in the context of earlier definitions of organisational modelling and the benefits of organisations in LSMASs. The concepts of OOVASIS, on the other hand, can be used to enrich the content of any MAS-related ontology or a model that works with ways of creating organisations in MASs, or that fosters cooperation of agents. One of the prominent benefits is observed in the testing and example development environment created by combining OOVASIS concepts with implementation-ready MAM5.

The combination of these ontologies, presented as the Lamrast−+ ontology, provides its user with a more expressive set of concepts for describing LSMASs. Such an ontology was published under the name of MAMbO5 [104], yet for the sake of consistency it is referred to as Lamrast−+ throughout this thesis. Apart from the basic organisational features of a system of agents, the new ontology features various other concepts that can be used for describing more complex LSMASs that feature human agents, location-dependent organisations, more detailed normative concepts, and a set of new or improved properties. On the other hand, the new ontology can, using its expanded or revised set of concepts, be used for creating a richer description of IVEs that feature organisational concepts, updated grouping concepts, and a revised take on normative concepts.

2.1.1.5 Activity Five: Implementation

> "The result of this phase is the ontology codified in a formal language [. . .]"
>
> — Fernández-López, Gómez-Pérez and Juristo [42]

The implementation activity is about completing the implementation process of the ontology. Codifying Lamrast−+ ontology in a formal language is performed using Protégé and OWL2. Both were chosen based on their widespread use in academic as well as in real sectors, and their status of formally defined or informally established standards in the context of ontology engineering. Class hierarchy created using Protégé is shown in Fig. 2.6.

Protégé [86] is the most widely used software for building and maintaining ontologies, with more than 250 000 users in 2015.

> "The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language

```
Declaration(Class(<OOVASIS_#OrganizationalUnit>))

EquivalentClasses(<OOVASIS#OrganizationalUnit>
ObjectUnionOf(<OOVASIS#Agent>
ObjectIntersectionOf(
    ObjectSomeValuesFrom(<OOVASIS#definesRoles> <OOVASIS#Role>)
    ObjectSomeValuesFrom(<OOVASIS#hasRelation> <OOVASIS#StructuralRelation>)
    ObjectSomeValuesFrom(<OOVASIS#hasRole> <OOVASIS#Role>)
    ObjectAllValuesFrom(<OOVASIS#hasRelation> <OOVASIS#StructuralRelation>)
    ObjectMinCardinality(1 <OOVASIS#definesRoles> <OOVASIS#Role>)
    ObjectExactCardinality(1 <OOVASIS#hasCriteriaOfOrganizing> <OOVASIS#CriteriaOfOrganizing>))))
SubClassOf(<OOVASIS#OrganizationalUnit> <MAM5#Agent>)
```

Listing 2.1: *OrganizationalUnit* concept rendered using OWL functional syntax

Figure 2.7: *OrganizationalUnit* concept relative to other ontology concepts

> for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. OWL 2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents."
>
> — W3C OWL Working Group [146]

The selected key concepts are covered in more detail as follows. The complete ontology rendered using OWL functional syntax is present in Appendix C.3.

The `OrganizationalUnit` concept, which plays a crucial role in the metamodel, is defined as presented in Listing 2.1, and is related to other concepts as shown in Fig. 2.7. This makes it take an intermittent position between the agent concept defined in JaCalIVE, and its more specified concept representing inhabitant agents, i.e. agents that can be represented physically.

The `Activity` concept is visualised (Fig. 2.8) using a complicated digraph, yet the definition (Listing 2.2) is not as complex – the activity concept is set to be equivalent to the concepts of `Behavior` (can be encountered in Smart Python Agent Development Environment (SPADE)-implemented systems) and `Agent_Action`.

The `Norm` concept is, by transition, defined as a subconcept in the context of both basic ontologies of this research (Fig. 2.9). A norm is thus positioned as an important organisational concept that can be specified as an `IVE_Law`. Functional OWL rendering of the `Norm` concept is provided in Listing 2.3.

```
Declaration(Class(<OOVASIS#Activity>))

AnnotationAssertion(rdfs:comment <OOVASIS#Activity> "Any atomic activity
    performed by some individual agent
")
EquivalentClasses(<OOVASIS#Activity> <OOVASIS#Behavior>)
EquivalentClasses(<OOVASIS#Activity> <OOVASIS#Behavior> <MAM5#
   Agent_Action>)
SubClassOf(<OOVASIS#Activity> <OOVASIS#Process>)
SubClassOf(<OOVASIS#Activity>
    ObjectIntersectionOf(
        ObjectMinCardinality(1 <OOVASIS#achieves> <OOVASIS#Objective>)
        ObjectExactCardinality(1 <OOVASIS#isPerformedBy> <OOVASIS#Agent>
           )))
```

Listing 2.2: *Activity* concept rendered using OWL functional syntax

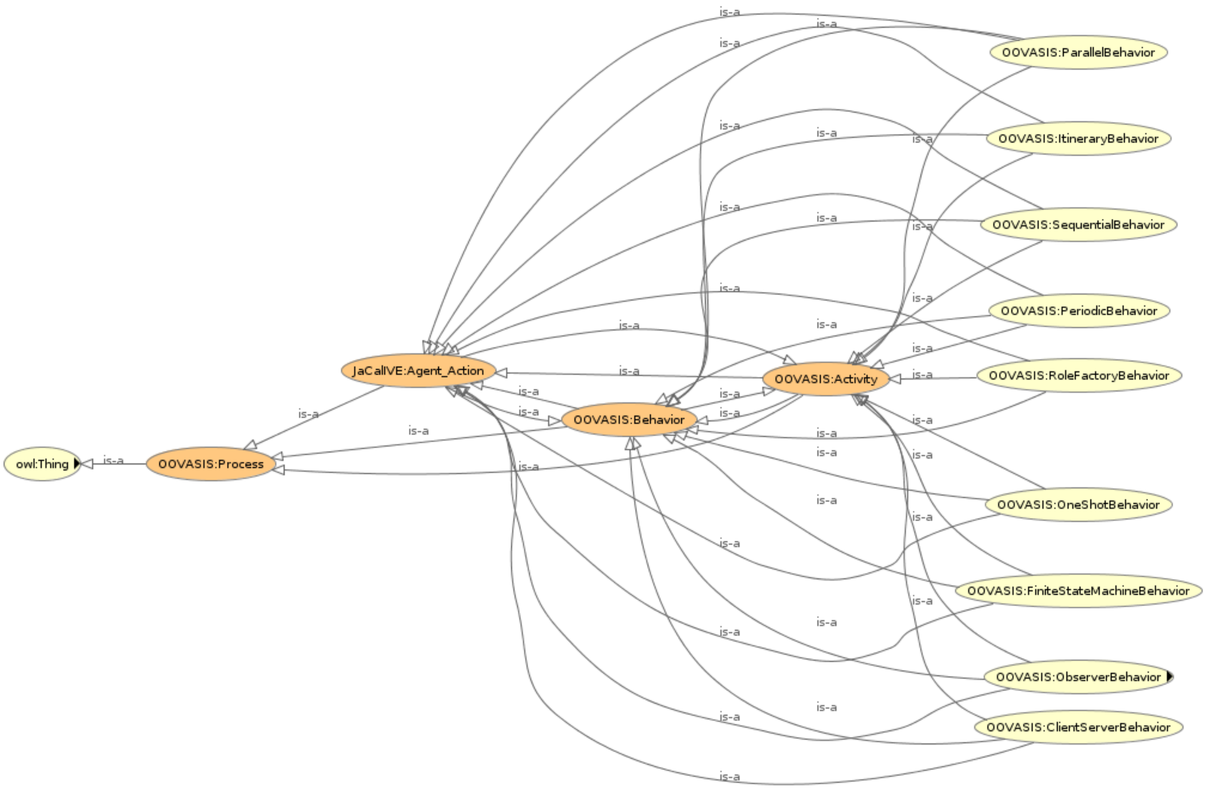


Figure 2.8: *Activity* concept relative to other ontology concepts

```
Declaration(Class(<OOVASIS#Norm>))

AnnotationAssertion(rdfs:comment <OOVASIS#Norm> "Norms are defined as (
   socially) accepted behavior in a defined group and represent a
   blueprint for behaving in said group")
SubClassOf(<OOVASIS#Norm> <OOVASIS#KnowledgeArtifact>)
```

Listing 2.3: *Norm* concept rendered using OWL functional syntax

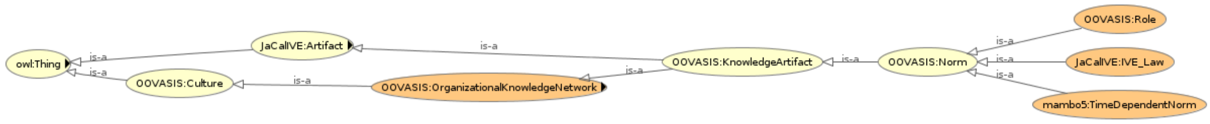


Figure 2.9: *Norm* concept relative to other ontology concepts

2.1.1.6 Activity Six: Evaluation

> "Evaluation means to carry out a technical judgement of the ontologies, their software environment and documentation with respect to a frame of reference (in our case the requirements specification document) . . . "
>
> — Fernández-López, Gómez-Pérez and Juristo [42]

To make sure the defined ontology is following some set rules of standard, and that it is ready to be used in the real world, the process of evaluation is necessary. Two segments of evaluation are verification and validation [42, 50]. Verification is the technical process that makes sure the designed ontology is correct in the context of associated software environments and documentation, with respect to a certain frame of reference. Validation, on the other hand, *guarantees that the ontologies, the software environment and documentation correspond the the system that they are supposed to represent.* [42]

In order to perform the **verification process** *we have to verify [the ontology's] architecture, its lexis and syntax, and its content.* [50]

Architecture verification is concerned with the structure of the developed ontology and how it follows *the principles of design of the environment in which the ontology is included.* [50] Lamrast−+ is defined using Protégé environment and is completely in accordance with the good practice examples associated with it. Furthermore, the designed ontology follows the basic guidelines of OWL2.

Lamrast−+ follows all the set rules of OWL2 and Resource Description Framework (RDF) as well, in the context of syntactic correctness and lexical structure. Defined classes are correctly defined as `owl:class` concepts, with all their associated properties defined accordingly. Formal definitions of the concepts included in Lamrast−+ are therefore verified for their lexical and syntactical correctness.

Content verification is the most complex component of the three stated, since it is:

> "[. . .] concerned with the analysis of completeness, consistency, conciseness, expandability, and robustness of the definitions and axioms that are explicitly stated in the ontology, and the inferences that can be drawn from those definitions and axioms."
>
> — Gómez-Pérez, Juristo and Pazos [50]

The importance of content verification is emphasised using the main goal of an ontology – knowledge reuse and sharing. Since the concepts are shared, and are expected to be further built upon and expanded, it is important to define concepts that guarantee such criteria to be met. Rules of Lamrast−+ ontology are set in such a way that the inference process is performed in its entirety and without unexpected results, proving positive consistency of the defined ontology, as no contradictory conclusions can be reached. Thus, there are no *contradictory sentences that may be inferred using other definitions and axioms.* [50]

Ontology completeness is a concept open for debate, but using the first activity of this methodology, laid out in Section 2.1.1.1, the scope of Lamrast−+ ontology is defined, and the ontology can be deemed to be complete in the context of this research, backed up by the following definition of semantically complete ontology:

> "– All that is supposed to be in the ontology is explicitly set out in it, or can be inferred using other definitions and axioms.
>
> – Each individual definition is complete."
>
> — Gómez-Pérez, Juristo and Pazos [50]

Lamrast−+ ontology is concise, for all the defined concepts are deemed useful and precise. Redundancies do exist (e.g. definitions of `Agent` or the concepts of `Behaviour`, `Activity`, and `Agent_Action`), but they do serve a purpose, namely to define a couple of synonymous concepts. The defined ontology can be used as a basis for further expansion. Indeed, the one of the intended expansions, intended to be performed as a part of a future research, is expanding the ontology with concepts that are specific to various LSMASs domains, such as MMORPGs, smart settlements, or similar.

When speaking of **validation**, the designed ontology has to be apt for representation of an intended system, and correspond to the elements of such a system, taking into account all the concepts of the ontology and the phenomena they are supposed to represent.

> "The validation of the ontologies against the frame of reference provides information about whether the ontology definitions are necessary and sufficient to represent the tasks and their solutions for different uses."
>
> — Gómez-Pérez, Juristo and Pazos [50]

The Lamrast−+ ontology can be used for modelling all the examples in Chapter 4. Since the models shown there are expressive enough, and they use only a set of selected concepts from the ontology, the ontology can be used to specify further details of the respective modelled systems, thus providing enhanced expressiveness, when compared to the metamodel, meaning that it can be used to further specify the appropriate systems.

2.2 Metamodelling

A model is an abstract representation of a real domain. In other words, *an abstraction of reality according to a certain conceptualization* [59, p. 31] referencing [54]. Fundamentally, it is an abstraction effort [64]. It is usually used to show a real-world phenomenon often in a simplified or stylised manner, therefore a *hypothetical description of a complex entity or process* [132, p. 14]. A model is defined in temporal terms as an approximation $M(t)$ of $S(t)$, where $S(t)$ is a system evolving over time [82]. A less formal or strict description of the concept of a model is given in [134]: *A model is a representation of something for someone's purpose somebody (sic) and developed by someone else.* meaning that every model *is author-driven and addressee-oriented, is aspect-related, is purpose specific, is limited in space, context and time, and is perspective.* An overview of various kinds of models is presented in [132].

The purpose of a model, being only a representation of a piece (i.e. an object or a phenomenon) of the real world, is that of analysis, observation, and research. Furthermore, a model is customised based on the goal that drives its creation, i.e. properties of a model depend on its purpose. For example a demographic model of inter-country migrations due to students attending various universities will not be useful for studying migratory behaviour of emigrants and immigrants on national basis. Another example may be a business process model where inclusion or omission of details of the model depend on the end-user of the model: lower management (such models include more detailed features of the observed system, such as stationery consumption and similar), or higher management (which is a model that omits the low-level nuances of an everyday life and presumably contains more abstract information that provides an overview of the business). Additionally, a model of a system comprising producers and consumers varies in details depending on the purpose of the model: analysis of goods transport, analysis of mergers and takeovers, overview of financial flow of an enterprise, etc. Modelling, as a method of constructing and describing models, is therefore conducted based on observation of a real-world situation, with the goal of creating an abstract tailored representation that can further be used for various purposes. Customisation of the model's features and their abstractness in representing the observed real domain is not intended to modify the observed situation, but to discern the features crucial for achieving the desired goal.

For all the former features, and many more, modelling is a method used extensively in science in general. Furthermore, digital models are gaining on popularity with the advancement of information sciences and computer performances.

Types of models are recognised based on a number of features, most prominent of which are [82]: time (static or dynamic), state (discrete or continuous), randomness (deterministic or stochastic), details and similarity (abstractness as a measure of repression of details, and fidelity as a measure of the real system's characteristics reflection), and dimensionality (dimensionality of representation – 1D to 3.5D). A different approach to types of models, in the context of information systems modelling is given in [148]: representation model, state tracking model, and system model.

In a manner similar to the transition of a real-life situation to a model, metamodelling introduces the model of a model, i.e. a metamodel. As mentioned before, a model is a simplified view on the given real domain. For example, real-life situation can include dozens of companies, and hundreds of consumers all of which interact with each other, buying and selling goods, demanding and providing services, forming coalitions etc. In a situation where observing organisational behaviour, in the context of organisational dynamics, of the involved companies is the primary goal, the model will be rather abstract, with minimal additional features being modelled, other than financial flows, spars messages, and basic account of demanded or provided services. The select few features are presumed to be enough for a satisfying analysis of merger behaviour of the observed companies. Each of the organisations and consumers, and other entities, is represented using a single element of the model, with identifying precision. Therefore, models are said to abstract information [132, p. 14]. Further discussion on levels of models, especially in the context of model-driven development, is provided by Atkinson and Kühne [6] and Muhanna and Pick [85].

The problem that may arise in the development of the described example model is the set of concepts to be used for representing the necessary elements of the model. Such a problem might not arise if the model is being built by a single person who is always around when the model is being referenced, but in case of a collaborative effort, the precise meaning of various elements might not be communicated clearly enough and misunderstanding could happen. Furthermore, if the model designer creates the given model, and comes back to it after a couple of years, they might have difficulties reading or further developing it. This is where metamodelling steps in.

A metamodel is in its core a model of a model – a definition of a set of concepts and their relationships [59]. The method of constructing models of models is therefore called metamodelling. Henderson-Sellers [59, p. 41], referencing [11], states that a metamodel is an *explicit specification of an abstraction expressed in a specific language.* Most of the time, a metamodel sacrifices domain specificity *for the benefit of reusability across domains.* [64] The relationship of a metamodel and a model is very similar to that of a model and a real domain, respectively. Whereas a model is an abstract representation of a real domain, a metamodel provides language elements for creating a model. The semantic nuances of the concept of a metamodel and relative view of the modelling levels surpassing sole model, are argued in [59], on the basis of the difference between metamodelling in the sense of M2 level (as opposed to modelling being M1), and the concept of metametamodel. An example of metamodelling levels is shown in Fig. 2.10, where `Computer game` is a part of a metamodel (level M2), with instances used in modelling on level M1 representing

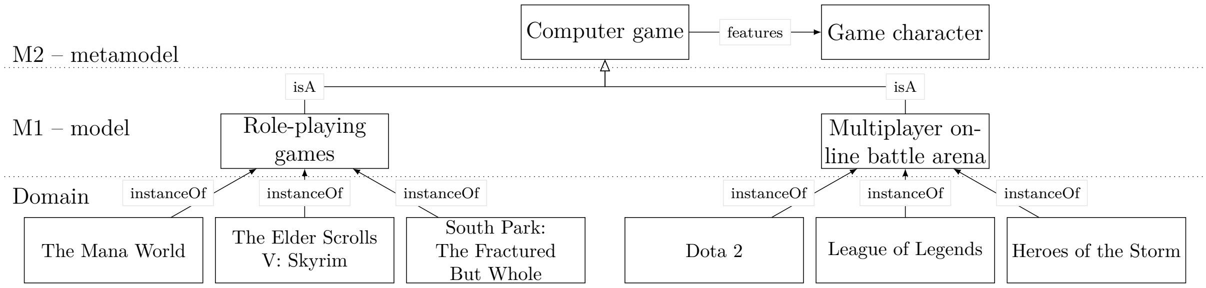


Figure 2.10: An example of metamodelling levels in the domain of computer games

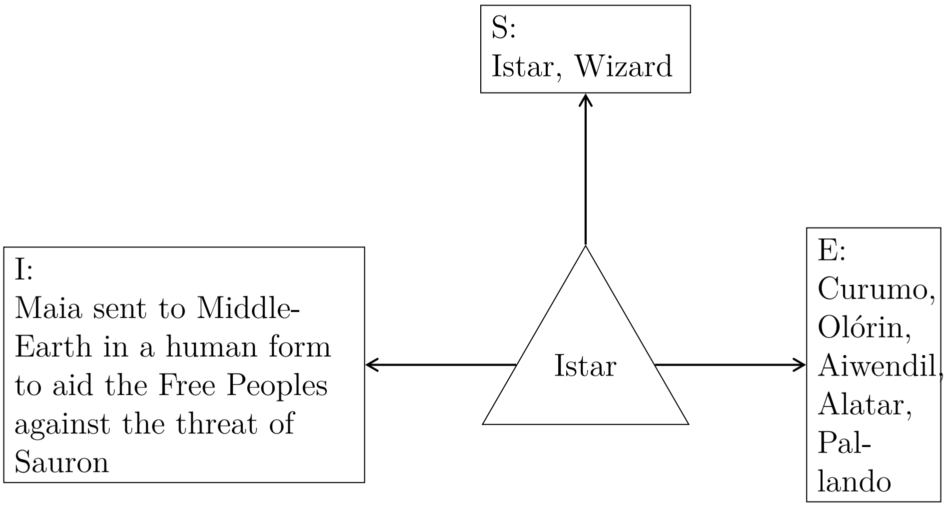


Figure 2.11: A specific concept, similar to [80]

concepts again, whereas instances are provided only on the lowest level designated as objects of the chosen domain. It may be argued whether instantiating is performed on a low enough level, but such an observation depends on the intended use of the model – for a purpose of tracking sold items and copyright infringement cases, it would be more useful if instances were particular instances of sold games, e.g. Ozano's Skyrim, Goran's League of Legends, Andrija's League of Legends, etc.

A short digression should be welcomed here, on the notion of a concept, in addition to what is mentioned in Section 2.1.1.3. By definition, a concept consists of three constituent elements: intension, extension, and a symbol. An intension is basically a definition of the concept, its description using features of the concept that define it for what it is, no more, no less, e.g. Maiar sent to Middle-Earth as human forms to aid the Free Peoples against the threat of Sauron. The extension includes all the instances of the given concept, e.g. Curumo, Olórin, Aiwendil, Alatar, and Pallando – the Istari from the lore of J.R.R. Tolkien. The symbol is a way of referencing the given concept, e.g. Istari or Wizards. Such a concept is visualised in Fig. 2.11. Early examples of the use of symbols and concepts to represent human thoughts are described on examples of ancient Egyptian hieroglyphs in [25].

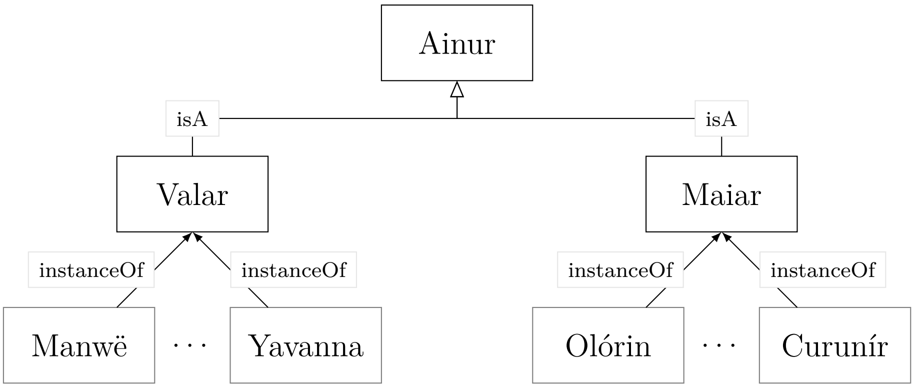


Figure 2.12: Concept hierarchy using *instanceOf* and *isA* relationships

A model therefore consists of elements that represent the modelled entities, i.e. extension consists of instances that represent real-world entities. A metamodel however consists of concepts the extension of which is populated by entities that are concepts themselves, i.e. concepts whose instances are concepts [105, p. 384]. The most prominent difference is therefore found in the distinction of two key relationships: *instanceOf* and *isA*. *instanceOf* is a relationship of a concept and an instance, while the *isA* connects two distinct concepts creating a hierarchy relationship between them with the meaning of one concept being a specific case of the other concept. One of the key criteria of their distinction is that, opposed to *instanceOf*, *isA* is transitive [105, p. 387].

A description based on two of the described examples follows. Istari are only a subset of the Maiar – they are the Maiar that are, as the intension defines, sent to Middle-Earth as human forms. Other Maiar include various other spirits that descended into Arda to help the Valar shape the World. The angel-like spirits of the Tolkien's legendarium[2] are divided into Valar (god-like beings) and Maiar (angel-like beings). Therefore, the concept of Ainur, representing all the spirits of Tolkien's legendarium, has the extension consisting of two concepts: Valar and Maiar. The described situation is shown in Fig. 2.12. Both of these concepts can further be observed, but in the context of naming the known beings, they do not represent particular individuals.

Metamodel is described by Kleppe [69] as *a model to specify language*. In the context of graph theory, a model is defined as a type graph with a set of constraints, i.e. *A model is a combination of a type graph and a set of constraints of various types.* [69]. A type graph is a mathematical construct that consists of a set of nodes and a set of edges between the nodes (each edge having a source and a target node), where nodes represent concepts, and edges represent relationships. An instance is thereafter described as a labelled graph in which every node and edge is of a type defined by the type graph, i.e. *An instance of a model M is a labelled graph that can be typed over the type graph of M and satisfies all the constraints in M's constraint set.* [69] Definitions of the selected

[2]Information of this example is based on `http://lotr.wikia.com/wiki/Valar`

concepts used herein are provided in Appendix B.1. Following the list of several types of possible constraints, the concept of metamodel is described in a manner similar to the descriptions and definitions in Section 2.2, yet with a refreshing addition: *A metamodel is a model used to specify a language.* [69] Constituent models of a language specification are an abstract syntax model, a concrete syntax model, and a semantic domain model.

2.2.1 Metamodelling Process

The process of defining a metamodel may be considered rather intuitive – the goal is to create a model that abstracts the given model of a real domain. Various steps of the metamodeling method have been defined by various authors [38, 134, 117, 110, 45], following their own interpretations of the metamodelling process therein. What follows is a short overview of a couple of views on the process of creating models, or applicable steps described in a similar fashion.

Four main dimensions of models are systematised as follows [134]: purpose, mapping, language, and value.

The purpose dimension of a model and modelling determines the reason a model is being defined, using intentions, goals, aims, and tasks identified as the goals to be solved by the model. Main concerns [134, p. 548] of this particular dimension are the impact of the model, the insight into the origin's properties, restrictions on applicability and validity, providing reasons for model value, and the description of how a model functions.

The mapping dimension is about the description of the modelled domain using the model, i.e. *a description of the solution provided by the model, the characterisation of the problem, phenomena, construction or application domain through the model.* [134, p. 547]

The language dimension is burdened with how to pick elements that allow the solution or the targeted domain to be expressed clearly. Some requirements can be defined [134] for language used in modelling or metamodelling, based on the established purpose of a model: means of representation, constructs, statements of relationship, scope, causal explanations, testable propositions, prescriptive statements.

The value dimension of a model is determined *by explicit statement of the internal and external qualities, and the quality of use* [134, p. 547]. The mentioned dimensions can be defined by keywords *wherefore*, *whereof*, *wherewith*, and *worthiness*, respectively.

A very important observation is given in [134, p. 547] about the dynamic nature of a model, as it is not an artefact that is *set in stone*, rather a concept of a changing nature, never being completed due to various sources of change, including scope insight, guiding rules, development plans, theories, mapping styles, etc.

Additional dimensions of models are emphasised in [134]: artefact dimension, user dimension, domain dimension, and context dimension. The mentioned dimensions are utilised in the description of the Lamrast−+ metamodel in the following sections.

A short overview of the metamodelling process is given in [117], in the context of metamodelling systems: listing properties that are required of particular metamodels being developed, and issues to be discussed and decided while creating a metamodel. Seven properties that are to be looked after include the purpose of the metamodel with sought after attention to the value of system output based on the values of system inputs, system optimisation, and similar; determining whether system responses are deterministic or random; how many variables are to be considered and whether they are qualitative or quantitative; what is the region of applicability and what is the amount of accuracy that is needed. It is evident that some of the properties given here encompass some of the dimensions of [134] described above. Since the metamodelling domain observed in [117] somewhat varies when compared to the domain of this thesis, decision issues are not discussed here in detail, excepting one, D5: *Does the metamodel have the necessary accuracy required?* that is discussed indirectly in Section 5.1 of this document.

A clear multi-step modelling cycle, based on observations in agent-based modelling domain, is proposed in [110, p. 7] referencing [51], since modelling may be observed as an iterative process [51], always improving the given model. The following tasks are presented, although it may not be necessary to perform the full cycle for every iteration, rather act in smaller cycles, as seen fit: formulate the question; assemble hypotheses for essential processes and structures; choose scales, entities, state variables, processes, and parameters; implement the model; analyse, test, and revise the model.

Formulating the question is the natural first step in the modelling cycle, demanding a clear research question to be formed and defined, since this research question is then used as a lead through the rest of the modelling cycle. Certainly, the posed question must not be too simple, or too complex, thus reformulation is a welcome method until the right research question is reached, one that is clear enough and achievable.

The second task is concentrated on formulating and assembling hypotheses concerning processes and structures essential to the problem addressed by the modelling process. Some of the questions regarding this task deal with identifying factors that have strong influence on the domain of interest, their mutual relationship and effects, and similar observations. This task is effectively a *brainstorming* session, generating hypotheses, but keeping in mind the necessity of simplification since the basic idea of the modelling cycle is to start with the most simple model possible, building up through cycle iterations.

A written formulation of the model is the result of the third task dealing with choosing scales, entities, state variables, processes, and parameters. Elements of the model are to be clearly described in detail in terms understandable by the model developer, and the intended user of the model.

Implementation task is charged with translating the verbal model description that was produced in the previous task into a model artefact. The model is thus produced using a computer software or other mean applicable to the given domain and model features.

The last task in the modelling cycle described by [51, 110] that is about analysing, testing and revising the implemented model, stimulates the model developer to learn from their model. Effectively, this task is the scene-setting process for the next iteration of the modelling cycle.

Some overlapping features can be observed between all the three modelling-related approaches described thus far, most notably clear definition of the purpose of the model, careful choosing of the elements of the model to be included, and ever-changing nature of a model.

Further study of the metamodelling process is continued here with observations from [45], although originally situated in the domain of building a simulation metamodel. A number of elements of the metamodel construction phase are proposed as follows: metamodel form proposal depending on the information uncovered during the target domain analysis; setting estimates of the parameters of the proposed metamodel as per simulation-generated data; metamodel verification conducted using various tests; metamodel validation derived from the simulation model validation performed by comparing it to actual data from the target domain or similar.

A sequence of six design steps is also presented in [45] in reference to [77], in the context of building a simulation model and metamodel: define the problem, define the ranges for the input variables, develop the experimental design, build a simulation model, develop the metamodel, validate the metamodel. Along with short descriptions of the mentioned design steps, some insight into metamodel validation process is given in [45], but the proposed validation is concentrated on simulation metamodels and is therefore omitted here.

The metamodelling process defined for this thesis consists of five activities: defining the level of abstraction, choosing concepts from the defined Lamrast−+ ontology, comparative analysis of the chosen concepts and existing approaches to large-scale multiagent systems (LSMASs) modelling, development of the metamodel, and its assessment. This sequence of activities is envisioned as a circular process for the sake of metamodel refinement. The stated sequence of activities is a customised summary of the meta- and modelling processes described in this section, further supported by analysing some other sources [157] and practical work during the research leading to this thesis.

2.2.1.1 Activity One: Level of Abstraction

This activity covers elements proposed as a part of the purpose dimension of a model [134], most of the question formulation and hypotheses assembling steps presented in [110, 51], and the problem definition steps of [45, 77].

The Lamrast−+ metamodel is set to deal with the problem of organisational modelling of LSMASs, with special emphasis on organisational dynamics. Organisational modelling of multiagent systems (MASs) is not in itself a state-of-the-art problem, since various approaches already exist (see Section 1.4.3), but application of such approaches to the domain of LSMASs is not utilised to a great extent. Furthermore, organisational dynamics is not a widely researched problem in the context of LSMASs, even though it is a concept that is of great interest in implementing LSMASs, as argued earlier, in Section 1.4. The problem tackled by this process is therefore definition of a metamodel that can be used for organisational modelling of LSMASs, with emphasis on organisational dynamics and application domain of massively multi-player online role-playing games (MMORPGs).

One of the research objectives of the research leading to this thesis is confronted by the Lamrast−+ metamodel: O2 Model organisational concepts applicable to MMORPGs.

Organisation is by many a definition a set of some entities, be it units, processes, etc. It is hard to talk about organisation when only a single individual is considered. MMORPGs recognise two main types of organisations or coalitions comprising various characters mostly representing players: parties and guilds. Main differences are temporal and membership-related.

Parties (a common name for this kind of an organisation in MMORPGs) are usually short-lived groups of players that are concentrated on accomplishing a set quest, without further attachments. Such organisations have simple structures, where the prominent criteria of organising is a common goal of the included agents (players). The number of members of a party is usually lower than that of a guild. The purpose of a party is to team up with (often unfamiliar) other players that share a quest or other driving goal with the party leader. Once the common goal is achieved, the party is usually disbanded. The described behaviour is commonplace, although deviations are allowed, e.g. friends cooperating in a game often play as members of the same party.

Guilds, as this type of an organisation is usually called, are by definition long-lived groups of players sharing more than a common quest. The criteria of organisation may be a strategic goal, the need for socialisation, etc. Guilds often develop internal organisation features, including hierarchical decision flows, coordinated event-attending activities, various organisation-related roles, etc.

When observing MMORPGs, the interesting organisations are those that are formed motivated by e.g. a hard quest. These party-level organisations tend to exhibit features of organisational dynamics most often and most prominently, out of all the organisation-related forms of cooperation between players of a MMORPG.

Since the emphasis of the Lamrast−+ metamodel is on modelling organisational features, agent-detailed modelling is of no interest. In other words, detailed modelling of an individual agent, i.e. an organisational unit, is of no concern to this metamodel, since the internal structure of an organisational units is not of grave importance. What is important are the actions that an organisational unit can perform, and to what extent can these actions be performed.

During the research leading to this thesis, it was concluded that the most suitable method of defining actions available to an agent is their definition as a part of a normative system. Therefore, the observed system should be defined using norms. As mentioned earlier in this thesis, when the Lamrast−+ ontology was considered, sets of norms included in a normative system are grouped into the concept of a role. Thus, a role, as a set of norms with a common denominator (DD A.37), defines which actions can be played by a specific organisational unit enacting the given role.

Even though details of an individual organisational unit do not have to be available for modelling using Lamrast−+ metamodel, an overly general approach is not welcome either. A metamodel that is too general might lead to a language with expressiveness problems, i.e. it may be unsuitable for a successful description of the real domain situation. The middle ground established during this research recognised the Lamrast−+ metamodel as being able to discern various kinds of organisational units (not necessarily individual instances of organisational units, but allowed if necessary), various roles available in the modelled system, actions defined by these roles, and goals achievable by the selected actions. All the elements should not be modelled in great detail, since one of the leading ideas of the metamodel is its implementation platform independence. Therefore, the language of the metamodel should not encourage implementation-specific values of a modelled system.

Following the described abstract-level-related characteristics of the Lamrast−+ metamodel, some additional features [134] are further provided below.

The finished Lamrast−+ metamodel is aimed at fostering modelling LSMASs with the emphasis on organisational features, especially dynamic changes in organisational features within the modelled system. As such, the metamodel should provide a viable solution for modelling LSMASs in one of their application domains, MMORPGs, by allowing the model developer to use concepts that are essential for modelling computer game-related situations and problems. These concepts are aimed at covering both the organisational and MMORPG domains.

The nature of MMORPGs was briefly described earlier in this thesis, with further details in the context of this research provided in [121, 120, 98, 97, 125, 124, 123, 138, 101, 127]. Individual players can advance through an MMORPG, yet their progress grows slower as they advance through the game. As the game advances, players can gain increased benefit from interacting with other players (in games that stimulate cooperative gameplay), and forming various types of groups of players (most prominent being parties and guilds, as described earlier here). Such coalitions or groups or organisations help individual players best the challenges they are faced with through the game. Furthermore, some in-game challenges are designed for larger numbers of organised players with a tactful approach. Additionally, MMORPGs usually have players playing characters of belonging to a single, a pair of, or a number of character classes – usually stereotyped character descriptors – warriors, archers, thieves, wizards, druids, etc. Depending on the class the character plays, different parts of the game are usable to the player, including varying gear, abilities, interactions, etc. MMORPGs are usually computer games that are quest-driven, i.e. game dynamics in the context of a story and campaign and game advancement is governed by in-game quests usually obtainable through interaction with non-player characters (NPCs) or special in-game events. These quests yield special rewards for their completion (e.g. special kind of loot, new quests, etc.). Some quests depend on the player's character being able to perform a specific in-game action or interaction, thus underlining the importance of character actions. The described view on the MMORPGs domain can be simplified and represented using the Lamrast−+ metamodel, in order to create an artefact that can be further used in the modelled system's development.

The Lamrast−+ metamodel can be therefore used when a quest-driven MMORPG world is to be described. Quest-driven feature is not a necessity, since goals can be defined, and quests are specialised goals by definition. The model has to be, certainly, modified if the game elements are modified, as the modelled system has to conform to the modelled properties of the observed system. Even though the primary application domain of Lamrast−+ metamodel are MMORPGs, it can be used to represent some other application domains of LSMASs, such as distributed sustainable systems, or other distributed systems comprising artificial intelligent agents.

The value of the Lamrast−+ metamodel stems from its wide suitable application domains, novelty inasmuch as it provides a simple language for modelling LSMASs and implementation of the modelled systems, comprehensibility found in the fact that only a numerically constrained set of concepts are defined that are easy to understand yet expressive enough for the possible challenges in modelling the primary application domain.

2.2.1.2 Activity Two: Choosing Concepts

The source of possible concepts to be included in the Lamrast−+ metamodel is the ontology described in Section 2.1, although some elements can be sourced to the domain-specific ontology presented in [98], since it provides concepts from the MMORPGs domain. The established level of abstraction in Section 2.2.1.1 governs the fact that not all concepts included in Lamrast−+ ontology are selected for inclusion in the Lamrast−+ metamodel. Therefore a short overview of the concepts deemed necessary to be included in the metamodel, and arguments in favour of such a decision, are presented in the following parts of this thesis. Only those concepts that have been selected for the set of concepts included in the metamodel are argued about here.

A set of rules that can be used to identify necessary objects among a large number of candidate objects and their respective properties is proposed in [148, 149]. The referenced set of rules is proposed in the context of ontology acting as a foundation for a metamodelling process and method engineering. More specifically, the set of rules in [148] has its source in object-oriented enterprise modelling. Out of the five main rules, the following are selected to be presented here:

> “
>
> (2) The candidate objects are those that represent things that become active (undergo external and possibly internal events) as a result of the interactions between the system and its environment.
>
> (3) The relevant properties of a thing that should appear in the model are only those that other objects must be "aware" of as a result of the interactions that propagate in the system. Thus, a certain attribute of a thing is modelled only if it is used or modified by other things (when the system interacts with the environment).
>
> (4) All information used in the system conveys states of objects. There is no "global" state information.
>
> ” — Wand [148]

Organisational Unit The main building block of an organisation is, as mentioned earlier in this thesis, an organisational unit. Here it represents induvidual agents, as well as groups of agents, following the recursive definition stated earlier in this thesis. The meaning of this is that organisational units should be considered in a rather abstract sense, i.e. as building blocks of an organisation, which is a view hinted at by the selected desired level of abstraction set up in Section 2.2.1.1. Therefore the more abstract concept of an organisational unit is favoured when compared to the concept of an agent, in addition to a rather obvious difference between these two concepts – an agent is always an agent, defined as either artificial or human, yet an organisational unit can be defined as a superconcept with agent as its subconcept, thus possibly containing more than agents. In the context of this metamodel, an organisational unit is a player’s character, hence indirectly a player, but groups of such individuals as well, and potentially groups of groups, etc. Based on everything stated here, organisational unit was deemed as a crucial element of the Lamrast−+ metamodel.

Role The concept of a role is a commonplace concept found in the domain of MMORPGs. A role defines a set of characteristics of the player’s character (be it an artificial player or a human player), and may have slight or great impact on the gameplay or the interaction and life of the character with other characters (players’ characters and non-player characters alike) within the game. A role can by definition include several varied types of normative constraints put on player characters, e.g. race, skill proficiency, class, etc. A role is therefore, when combined with the definition of a role, unavoidable concept when normative systems are observed. A role in the

Lamrast−+ metamodel represents a set of norms of the system that are applicable to organisational units within the system. Furthermore, as units of constraint, roles make certain actions available to be performed by organisational units playing particular roles. A role of a wizard thus allows the character to performs spells. Roles can be much more specific than e.g. classes are in MMORPGs, or can be completely unrelated. The concept of a role is best described by the model designed, and are not necessarily a direct copy of the groups of constraints in the original observed system. Deriving from all the stated here, the role concept is an important companion of the concept of an organisational unit, and an important element of a normative system.

Action An agent affects its environment by using actions at its disposal. Actions are therefore the mode of interaction between an agent and its environment. Every action can thus be defined as having a source and a target states, with a defined action in between. Since an action can be described as a metaphor for a piece of any and all interaction between an agent and its environment, i.e. between a character and the rest of the in-game world, it is recognised as an important element of an organisational metamodel of LSMASs. Furthermore, an action is the key middle element when organisational units playing a role are faced with fulfilling a set objective – the action necessary for fulfilling a given objective can be performed by a single role instance, when it is enacted by a given organisational unit. Several different actions can be grouped into a process, yet their relationship is different than the recursive one found when organisational units are considered.

Objective The concept of an objective was selected as one of the concepts of Lamrast−+ ontology, all of them having a similar meaning pertaining to an almost identical real-life object. Based on the selected level of abstraction described in Section 2.2.1.1, the concept of an objective was chosen in favour of any of the other concepts of similar meaning in the Lamrast−+ ontology. Although the MMORPGs domain is most comfortable with the concept of quests, it is argued that the concepts of both a quest and a goal are not adequate since a quest is most often a series of tasks, and a goal is short-termed or not timed by definition. Contrariwise, an objective is said to be presenting a more generalised view on the matter, featuring more extensive use cases. Even though it is by definition a concept encompassing both the concept of a quest and that of a goal, it can be used as a mix of the two in the metamodel, best described later in the thesis.

Knowledge Artefact Even though it may not be a key concept when an organisation is considered, and especially when the MMORPGs domain is used as a pretext, the knowledge artefact concept is a welcome addition to the set of concepts describing normative LSMASs. A knowledge artefact is described as storing agent knowledge

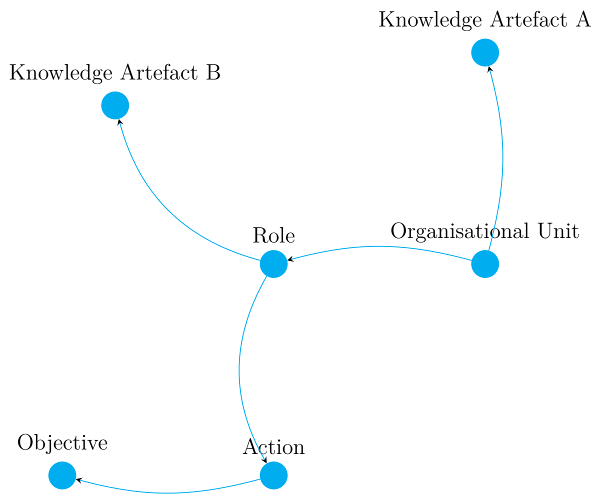


Figure 2.13: Visualised concepts of the metamodel and their non-detailed properties

relevant to the agents of the given system. Since agent knowledge is not easy to be described using generally applicable methods, a knowledge artefact is a formalised representation of a piece of knowledge relevant to the agents within the observed system. Descending to a more detailed observation of the knowledge artefact concept, two distinct but related concepts are identified: individual knowledge artefact and an organisational knowledge artefact. Individual knowledge artefacts are defined as knowledge pertaining to individual organisational units (e.g. character characteristics, their skills, and similar individual-level data), while organisational knowledge artefacts contain knowledge that should be available system-wide since it contains norms, guidelines, system-level knowledge, etc.

The above concepts are those most interesting and worthwhile for an organisational metamodel. Arguably, some of the concepts of Lamrast−+ ontology are more suitable to be observed as concepts for describing an organisation, as opposed to modelling features of an organisation, such as the subclasses of organisational structure. It is useful to note here that the concepts of the Lamrast−+ metamodel are supposed to be used to describe a system to be implemented, whereas the organisational developments within the system are a matter of the system, and not entirely of the model of that system. In other words, run-time organisations and their features depend on the implementation details of the system, such as agent-based details, permitted actions and interactions, and similar.

In addition to the concepts described above (Fig. 2.13), their properties have to be defined for the metamodel to be defined clearly.

The decision not to include most of the concepts encountered within the ontology is motivated by the goal of creating a simple yet expressive metamodel.

It is clear from the provided argumentation above that some quite important concepts were left out from the Lamrast−+ metamodel, such as types of organisational structure. The main reason for such a decision is that, with the purpose of the Lamrast−+ metamodel in mind, along with some of its other features, some of the concepts of the ontology would demand an overly specific metamodel, which is not in accordance with the prescribed abstraction guidelines for this metamodel.

2.2.1.3 Activity Three: Comparative Analysis

An analysis of existing models that can be used for modelling organisations in MASs domain, although only some of them are intended for LSMAS domain, in the context of this research, is given in [92], where the existing models, described in some detail in this thesis under Section 1.4.3, are put in context regarding the Lamrast−+ metamodel. Descriptions given in Section 1.4.3 are given with respect to Lamrast−+ metamodel as well.

Some common features of these models can be derived from their descriptions in Section 1.4.3: individual units are always in the spotlight, along with normative elements translatable to roles. Since a MAS is about interaction of agents and their impact on the system environment, a concept of action, detailed to some extent, is always present in a model dealing with the concept of organisation in MASs.

The selected key concepts described in Section 2.2.1.2, combined and utilised as parts of the Lamrast−+ metamodel bring additional value not present in the models mentioned in Section 1.4.3, as far as this research is considered. Furthermore, it should be noted here that existing models either deal with MASs primarily, thus leaving modern needs of distributed systems describable as LSMASs wanting, or have had their development stopped at the stage of meta- or model descriptions, without clear use-cases or tools that can be utilised for using the developed meta- or models. Lamrast−+ metamodel is by default intended to be used within the context of LSMASs, which is a point of view backed up by the metamodel conforming to the modern perspectives of organisational modelling for LSMASs presented in [118]. Additionally, Lamrast−+ metamodel is provided not only theoretically, but as a practically usable artefact as well.

Out of all the models presented in Section 1.4.3, Lamrast−+ metamodel is compared here with NOSHAPE MAS organisational model described by Abbas [1]. NOSHAPE MAS is chosen as the model Lamrast−+ metamodel is compared to since it is intended for organisational modelling of large-scale systems comprising agents grouped on several levels of abstraction, it is similar in offered concepts to Lamrast−+ metamodel, and belongs to recently published research (published in 2014). Comparative description is provided in Table 2.2, where the used customised criteria is mostly based on concepts featured in either of the metamodels, and in general coordination with earlier mentioned perspectives of organisational modelling of LSMASs [118].

Table 2.2: Comparative description of Lamrast−+ metamodel and NOSHAPE MAS [1]

	Lamrast−+	**NOSPAHE MAS**
Organisation	Organisation is defined as a set of organisational units organised using a specific criteria of organising. The element of an organisational unit can be used to represent an individual agent or a group of agents. The amount of modelled organisational levels is virtually unlimited, as shown in Chapter 4.	Three levels of abstraction are defined, where `Organisation` is considered the lowest, with `World` and `Universe` defined above it. Although most of the features of these three concepts are shared amongst them, the principle difference is of semantic value.
Agents	There is no concept in Lamrast−+ metamodel that should be exclusively used for modelling individual agents. Agents are modelled as organisational units, since every MASs can be considered an organisation, with various organisational features defined to an extent.	Agents are considered lower-level entities that enact roles, execute tasks, and use various resources (e.g. databases).
Roles	A role is considered as a set of organisational norms and is meant to be enacted by organisational units, thus making defined actions available to them. On the meta- and model no distinction is made between types of roles – they are all modelled in the same way. Roles can be enacted by organisational units, regardless of their atomicity, i.e. individual or compound organisational unit.	Roles are defined by organisations and can be played by agents. On the metamodel level two role types can be distinguished: static (specific to each organisation and concerned with its structural features) and dynamic (domain specific, can be shared, exchanged, and moved between organisations).

Continued on next page

Table 2.2 – continued from previous page

	Lamrast−+	NOSPAHE MAS
Actions	Actions can be modelled and related to roles. Based on the roles they are associated with, actions can be performed by organisational units, depending on the role played by the given organisational unit. Actions can be modelled as a part of a process. Each action can be used to achieve an associated objective.	While actions are not defined explicitly, agents are modelled as entities that can execute `Tasks`, and roles can utilise `Interaction Protocols`. These concepts can be, in part, considered as concepts that correspond with the concept of `Action` in Lamrast−+ metamodel.
Strategy	Both `Objective` and `Process` concepts can be used to define aspects of an organisation's strategy, with objectives being a more prominent example. Objectives can be modelled as simple (atomic objectives, i.e. being achievable by a single action) or complex (consisting of non-predefined levels of sub-objectives, the lowest of which are simple or atomic objectives).	No elements exist for modelling strategy-related concepts.
Tools	The modelling tools is an integral part of Lamrast−+ metamodel, as it renders the metamodel usable. The added feature of implementation template generation is a benefit none of the models possess, as far as the author is aware.	No apparent tools are presented in the original research, nor in subsequent research, as far as the author of this thesis is aware.

End of Table 2.2

Further discussion on Lamrast−+ metamodel and its evaluation is presented in Section 5.1.

2.2.1.4 Activity Four: Metamodel Development

The metamodel development process was performed in cycles, where each cycle was used to add, remove, or modify various concepts included in the metamodel, upon those described in Section 2.1.1.2. Some concepts were added for ease of metamodel implementation, especially because they were identified as having properties common to more than one concept.

Overview of the finished Lamrast−+ metamodel is presented in Fig. 2.14, with all the classes representing the included concepts and their relationships.

The overview of the metamodel, provided in Fig. 2.14, shows graphically the noted significance and centrality of the concepts of organisational unit and role.

Associated classes are represented as connected using a solid arrow, while inheritance is shown as classes connected via a solid arrow with a hollow arrow head. Concepts are shown as rectangles, and their properties (i.e. their relationships) are shown having hexagonal headers.

The concept of organisational dynamics that is emphasised throughout this thesis is dealt with using graph grammars, since the whole metamodel, and the model created using it, is in form of a graph. For the purpose of this thesis, an introduction to graph grammars, with key definitions, is provided in Appendix B.2. Rules were created that can be applied to modifying the model created using the metamodel accordingly. These rules either describe situations with an organisational unit playing a certain role and using its actions to create new organisations, or to describe situations where organisational units can join other organisational units (i.e. organisations). The initial mention of Lamrast−+ metamodel's graph grammars and their role in representing organisational dynamics is given in [125]. Further description of graph grammars, and their use in this metamodel, with examples, is shown in Section 2.2.2.

Pages 64 to 69 contain a detailed account on the developed metamodel elements, and their associations, with description provided where deemed convenient. Details of class concepts are provided first, followed by associations. Concept descriptions are not provided here, since they are defined in other parts of this thesis (e.g. Section 2.2.1.2). Before element-based details are provided, it should be noted here that all the elements of Lamrast−+ metamodel have attribute `ID` which is used for unique identification of model elements.

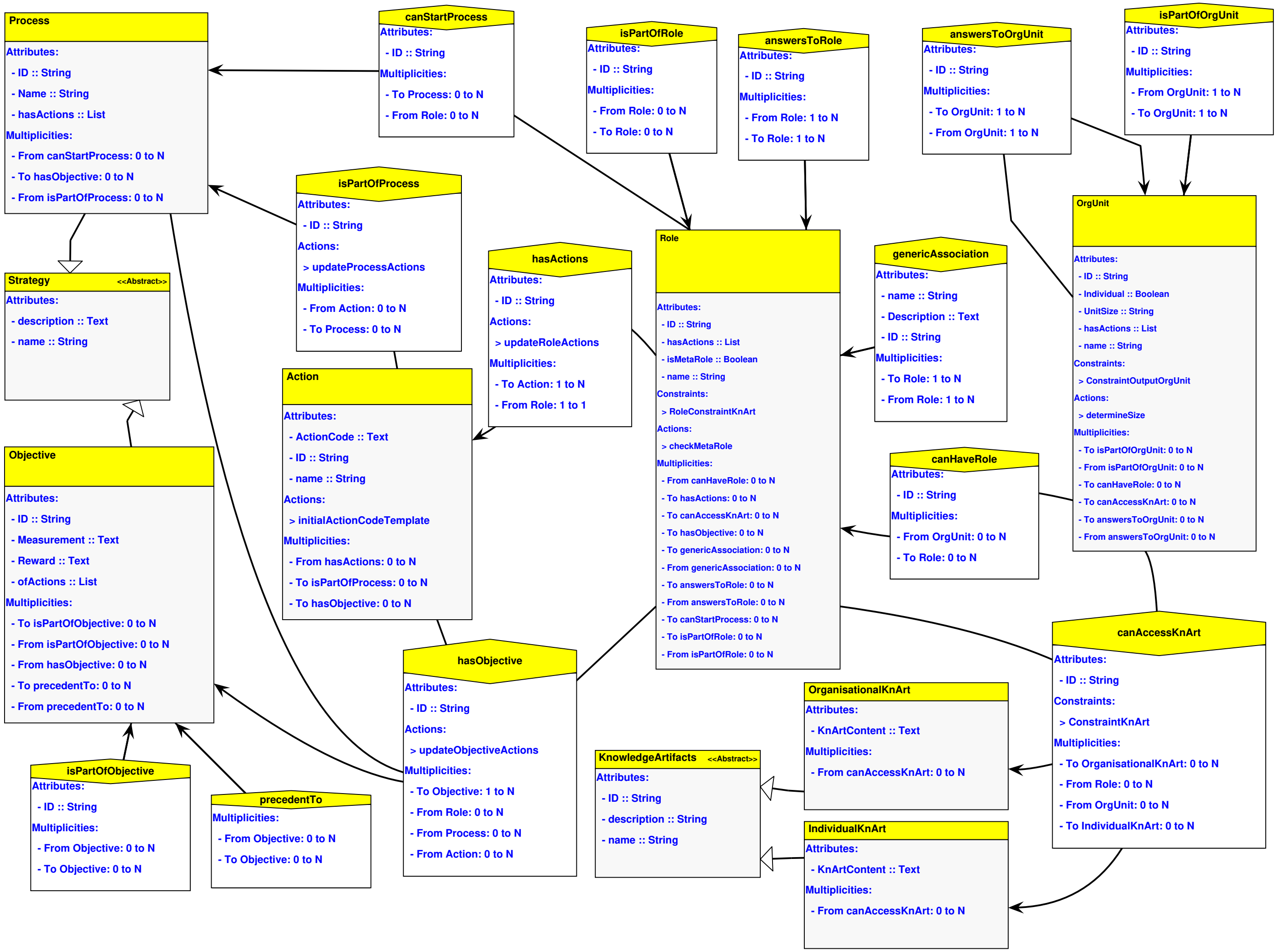


Figure 2.14: Overview of the Lamrast−+ metamodel

Concept `Role`

– **attributes** Each `Role` individual is defined by the following attributes:

- `ID` is used for unique identification of model elements.
- `hasActions` is a list of actions that the given `Role` is associated with, used for storage purposes and a quick overview of connected actions in graphic view.
- `isMetaRole` is a boolean value that defines whether the given role has some sub-roles defined.
- `name` contains the name of the role to be displayed in graphic view and used when implementation template is generated.

– **constraints** There is only one active `Role` constraint.

- `RoleConstraintKnArt` is a constraint which defines that a `Role` individual can only be connected to an `OrganisationalKnArt` individual. This constraint is necessary since a shared relation type is defined for connecting `Role` and `OrgUnit` concepts to knowledge artefact concepts.

– **actions** Only one action is defined for `Role` concept.

- `checkMetaRole` is an action that is run when a `Role` individual is connected to another `Role` individual, and is used to set the `isMetaRole` attribute value automatically.

– **connections** Each `Role` individual can be connected to the following concepts:

- `OrgUnit` connected to a `Role` means that the given `OrgUnit` individual can enact any of the connected `Role` individuals. The logical nature of the connection depends on the graphical representation of the connection, since an organisational unit cannot play all the available roles at the same time.
- `Action` individual related to a `Role` individual describes an action that is made available to an organisational unit when it enacts the given role.
- `OrganisationalKnArt` individuals store organisational knowledge and are therefore made available when a role is enacted by an organisational unit.
- `Process` individual related to a `Role` individual is a case that does not show up often, and is serves simply to show a group of related actions that are available to a `Role` individual.
- `Objective` is not used.

Concept `OrgUnit`

– **attributes** Each `OrgUnit` individual is defined by the following attributes:

- `ID` is used for unique identification of model elements.
- `Individual` is a boolean value type attribute that defines whether the `OrgUnit` individual represents an individual organisational unit or a group of individuals.
- `UnitSize` serves the same purpose as attribute `Individual`, but this one is further used in graphic view of a model.
- `hasActions` is a list of actions that are defined on an organisational unit level, i.e. they do not depend on the role enacted by an organisational unit.
- `name` contains the name of the organisational unit to be displayed in graphic view and that is to be used when implementation template is generated.

– **constraints** There is only one active `OrgUnit` constraint.

- `ConstraintOutputOrgUnit` is a constraint which defines that an `OrgUnit` individual can only be connected to a single knowledge artefact individual.

– **actions** Only one action is defined for `OrgUnit` concept.

- `determineSize` is an action that is run when an `OrgUnit` individual is connected to another `OrgUnit` individual, and is used to set the `Individual` and `UnitSize` attribute values automatically.

– **connections** Each `Role` individual can be connected to the following concepts:

- `Role` connected to an `OrgUnit` individual describes that the given `OrgUnit` individual can enact any of the connected `Role` individuals. The logical nature of the connection depends on the graphical representation of the connection, since an organisational unit cannot play all the available roles at the same time.
- `IndividualKnArt` individuals store individual knowledge and are therefore made available to an organisational unit.

Concept `Action`

– **attributes** Each `Action` individual is defined by the following attributes:

- `ID` is used for unique identification of model elements.

- `ActionCode` string attribute contains implementation template necessary for implementing agent action using a chosen LSMASs development environment. The only available option at the moment is Smart Python Agent Development Environment (SPADE), and the associated implementation template for agent behaviours. The code input here is copied into the generated implementation template feature provided by the accompanying modelling tool.
- `name` contains the name of the action to be displayed in graphic view and that is to be used when implementation template is generated.

– **actions** Only one action is defined for `Action` concept.

- `initialActionCodeTemplate` is an action that is run when an `Action` individual is created or edited, and is used for setting up the `ActionCode` attribute value, i.e. for generating implementation template for the given agent action.

– **connections** Each `Action` individual can be connected to the following concepts:

- `Role` connected to an `Action` individual describes that the given `Role` individual can conduct the specific action, and makes that particular action available to the `OrgUnit` individual that chooses to enact the given role.
- `Process` individuals, when `Action` individuals are connected to them, represent a grouping concept, i.e. a set of actions that, when used in a combination, can achieve a set objective.
- `Objective` is a concept denoting to what end can an action be used, i.e. what is the intended result of performing a specific action.

Concept `Process`

– **attributes** Each `Process` individual is defined by the following attributes:

- `ID` is used for unique identification of model elements.
- `hasActions` is the list of `Action` individuals connected to the given `Process` individual, denoting all the actions that are considered a part of the given process.
- `Name` deprecated – was replaced with `name`.
- `name` contains the name of the process to be displayed in graphic view and that is to be used when implementation template is generated. Inherited from `Strategy`.

– **connections** Each `Process` individual can be connected to the following concepts:

- `Role` connected to a `Process` individual describes that the given `Role` individual can conduct the specific process, having already defined available `Action` individuals as well.
- `Action` individuals, when connected to `Process` individuals, represent parts of a group, i.e. a set of actions that, when used in a combination, can achieve a set objective.
- `Objective` is a concept denoting to what end can a process be used, i.e. what is the intended result of performing a specific process.
- `Strategy` is an abstract concept that is not intended to be instantiated, as it serves as a generalised concept of both `Process` and `Objective` concepts, and their common attributes.

Concept `Objective`

– **attributes** Each `Objective` individual is defined by the following attributes:

- `ID` is used for unique identification of model elements.
- `Measurement` is the mechanism that can be used for measuring when an objective is achieved, i.e. what is the state of the in-game world that has to be achieved for the objective to be considered fulfilled. This feature is not yet implemented as a part of the application template generator.
- `Reward` received by the agent who successfully solves this particular objective is defined here. This feature is not yet implemented as a part of the application template generator.
- `ofActions` is a list of actions that a particular `Objective` individual is connected to.
- `name` contains the name of the objective to be displayed in graphic view and that is to be used when implementation template is generated. Inherited from `Strategy`.

– **connections** Each `Objective` individual can be connected to the following concepts:

- `Role` is not used.
- `Action` individuals, when connected to `Objective` individuals, represent to what end can an action be used, i.e. what is the intended result of performing a specific action.
- `Process` individuals, when connected to `Objective` individuals, represent to what end can a process be used, i.e. what is the intended result of performing a specific process.

- `Strategy` is an abstract concept that is not intended to be instantiated, as it serves as a generalised concept containing both `Process` and `Objective` concepts, and their common attributes.

Concept `Strategy`

– **attributes** Each `Strategy` individual is defined by the following attributes:

- `description` is a textual attribute containing natural language description of the given `Strategy` individual, i.e. a `Process` or an `Objective`.
- `name` contains the name of the `Strategy` individual to be displayed in graphic view and that is to be used when implementation template is generated. Inherited by `Process` and `Objective`.

Concept `OrganisationalKnArt`

– **attributes** Each `OrganisationalKnArt` individual is defined by the following attributes:

- `ID` is used for unique identification of model elements. Inherited as such from `KnowledgeArtifacts`.
- `description` is a textual attribute containing natural language description of the given knowledge artefact individual. This feature is not yet implemented as a part of the application template generator. Inherited from `KnowledgeArtifacts`.
- `KnArtContent` is a textual attribute that contains the content of the given knowledge artefact. Since agent knowledge is for the purposes of this research explicated using Prolog, the contents of this attribute should be defined using Prolog as well. This feature is not yet implemented as a part of the application template generator.
- `name` contains the name of the objective to be displayed in graphic view and that is to be used when implementation template is generated. Inherited from `KnowledgeArtifacts`.

– **connections** Each `OrganisationalKnArt` individual can be connected to the following concepts:

- `Role` individual connected to an `OrganisationalKnArt` individual denotes organisational knowledge available to a specific role. Knowledge associated with a specific role is made available to an organisational unit when it enacts the given role.
- `OrgUnit` individuals cannot be connected to an `OrganisationalKnArt` individual.

- `KnowledgeArtifacts` is an abstract concept that is not intended to be instantiated, as it serves as a generalised concept containing both `OrganisationalKnArt` and `IndividualKnArt` concepts, and their common attributes.

Concept `IndividualKnArt`

– **attributes** Each `IndividualKnArt` individual is defined by the following attributes:

- `ID` is used for unique identification of model elements. Inherited as such from `KnowledgeArtifacts`.
- `description` is a textual attribute containing natural language description of the given knowledge artefact individual. This feature is not yet implemented as a part of the application template generator. Inherited from `KnowledgeArtifacts`.
- `KnArtContent` is a textual attribute that contains the content of the given knowledge artefact. Since agent knowledge is for the purposes of this research explicated using Prolog, the contents of this attribute should be defined using Prolog as well. This feature is not yet implemented as a part of the application template generator.
- `name` contains the name of the objective to be displayed in graphic view and that is to be used when implementation template is generated. Inherited from `KnowledgeArtifacts`.

– **connections** Each `IndividualKnArt` individual can be connected to the following concepts:

- `Role` individuals cannot be connected to an `IndividualKnArt` individual.
- `OrgUnit` individual connected to an `IndividualKnArt` individual denotes individual knowledge available to a specific organisational unit.
- `KnowledgeArtifacts` is an abstract concept that is not intended to be instantiated, as it serves as a generalised concept containing both `OrganisationalKnArt` and `IndividualKnArt` concepts, and their common attributes.

Concept `KnowledgeArtifacts`

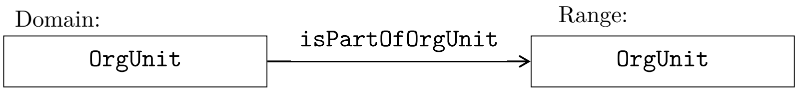
– **attributes** Each `KnowledgeArtifacts` individual is defined by the following attributes:

- `ID` is used for unique identification of model elements.
- `description` is a textual attribute containing natural language description of the given `KnowledgeArtifacts` individual, i.e. an `OrganisationalKnArt` or an `IndividualKnArt`.

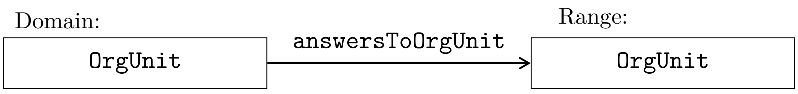
- `name` contains the name of the `KnowledgeArtifacts` individual to be displayed in graphic view and that is to be used when implementation template is generated. Inherited by `OrganisationalKnArt` and `IndividualKnArt`.

After the class concepts are described above, the other important type of elements in the Lamrast−+ metamodel should be described – associations. Both of these element types can be seen in Fig. 2.14. While concepts (classes) are shown as rectangles, associations are visually represented as rectangles with hexagons on top. The main difference between these two types of metamodel elements is evident in modelling using this metamodel. Concepts are instantiated as objects and associations are instantiated as relations between those objects. Therefore, for example, the association `canHaveRole` that connects organisational units with roles they can enact will be visualised in a system's model as a relation between an `OrgUnit` individual and a `Role` individual. By default, associations of Lamrast−+ metamodel contain only one attribute – `ID` – which is used for unique identification of model elements. Pages 70 to 72 therefore provide the overview of association elements without stating the single and default `ID` attribute.

Association `isPartOfOrgUnit` can be created between two `OrgUnit` individuals, and denotes that an organisational unit is a part of another organisational unit, thus building the idea of a higher-level (compound) organisational unit comprising lower-level (more simple) organisational units.

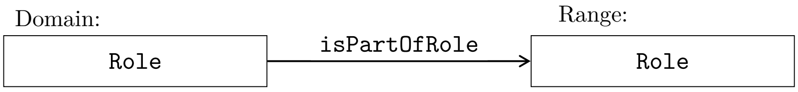


Association `answersToOrgUnit` can be created between two `OrgUnit` individuals, and denotes that an organisational unit is located on a hierarchically lower level, when compared to the associated organisational unit. This feature is not yet implemented as a part of the application template generator.

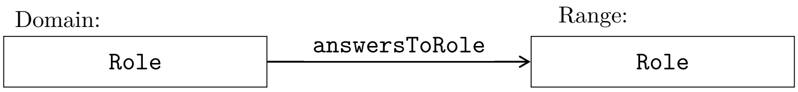


Association `canHaveRole` can be created between an `OrgUnit` individual and a `Role` individual, and denotes that an organisational unit can enact a certain role. Logic of the relation is dealt with in graphic view, when a model is being defined – `Role` individuals connected to an `OrgUnit` individual using a single `canHaveRole` association element cannot be enacted simulatneously, i.e. only one role can be enacted by a given organisational unit per `canHaveRole` association defined. This feature is not yet implemented as a part of the application template generator.

Association `isPartOfRole` can be created between two `Role` individuals, and denotes that a role is a part of another role, thus building the idea of a higher-level (complex) roles comprising lower-level (more simple) roles. The purpose of this association is to define a single higher-level role that a system modeller could then connect to an `OrgUnit` individual, thus making all the lower-level roles available to the given organisational unit. This feature is not yet implemented as a part of the application template generator.

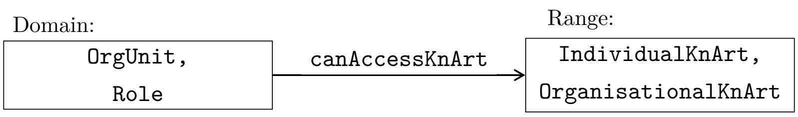


Association `answersToRole` can be created between two `Role` individuals, and denotes that a role is located on a hierarchically lower level, when compared to the associated role. This feature is not yet implemented as a part of the application template generator.

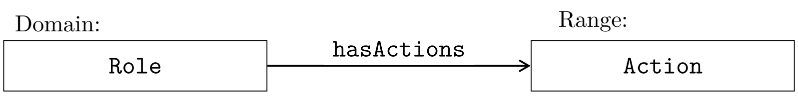


Association `genericAssociation` can be created between two `Role` individuals, with the role of a placeholder, for defining an association that is not defined by default. This feature is not yet implemented as a part of the application template generator, although it can be used in model development.

Association `canAccessKnArt` can be created between an `OrgUnit` or a `Role` individual, and an `IndividualKnArt` or an `OrganisationalKnArt` individual, respectively. Individual knowledge artefacts (containing knowledge about a given agent's individual features) are available to organisational units only, while organisational knowledge artefacts (containing pieces of organisational knowledge, organisational culture, etc.) are available to roles only.



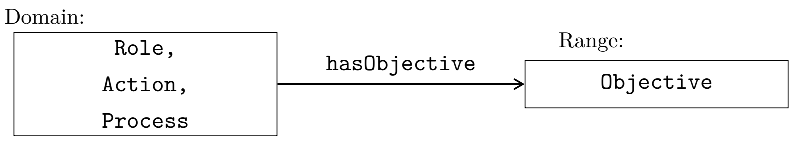
Association `hasActions` can be created between a `Role` individual and `Action` individuals, and denotes actions that are available to a given role. When an organisational unit enacts a particular role, the associated actions are made available to it.



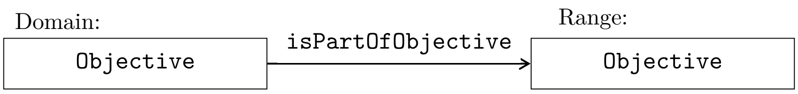
Association `canStartProcess` can be created between a `Role` individual and an `Action` individual, and denotes a process that can be started by a given role, where a process is built only of actions available to the given role. This feature is not yet implemented as a part of the application template generator.

Association `isPartOfProcess` can be created between an `Action` individual and a `Process` individual, and denotes that an action is a part of a process which serves as a grouping concept. A process is here defined as a set of actions that can be used in unison to achieve a set objective. This feature is not yet implemented as a part of the application template generator, since actions are defined atomary and with their own objectives.

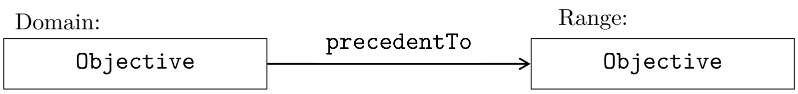
Association `hasObjective` can be created between either a `Role` or an `Action` or a `Process` individual and an `Objective` individual, and denotes that a role, an action, or a process have a specific objective, i.e. that they strive to, or can be used to achieve, respectively, a specific state of the system (or an in-game world in the context of MMORPGs). This feature is partially implemented as a part of the application template generator.



Association `isPartOfObjective` can be created between two `Objective` individuals, and denotes that an objective is a part of another objective, i.e. that an objective consists of a set of objectives, namely that an `Objective` individual is a part of a complex objective. This does not denote, of course, that the lower-level objective is atomic.



Association `precedentTo` can be created between two `Objective` individuals, and denotes that an objective precedes another objective, i.e. that it is advised to achieve one objective before achieving the other.



2.2.1.5 Activity Five: Metamodel Assessment

This metamodel is created with seven perspectives of organisational modelling of LSMASs in mind, presented in [118]:

- organisational structure (decision and information flows of an organisation),
- organisational culture (important intangible aspects of an organisation including knowledge, norms, reward systems, language and similar),
- strategy (long term objectives of an organisation, action plans for their realisation as well as tools on how to measure success),
- processes (activities and procedures of an organisation),
- individual agents (the most important asset of any organisation – individual agents actually performing the work),
- organisational dynamics (organisational changes including reorganisation of any of the mentioned components),
- context and inter-organisational aspects (organisational behaviour towards its environment including strategic alliances, joint ventures, mergers, splits, spinouts, and similar)

The metamodel assessment activity was envisioned as an evaluation process comparing the features of the metamodel with the above perspectives of organisational modelling of LSMASs, since they represent a modern approach to modelling LSMASs, and the metamodel should be capable of modelling modern applications of the LSMASs domain, including MMORPGs.

The Lamrast−+ metamodel allows the model developer to define various roles and organisational units, and relations between them that define decision flows e.g. `answersToRole` which is a typically hierarchical relation. However, organisational structure does not have to be hierarchical, which is why roles and organisational units can be defined independently of each other. Another feature towards fulfilling this perspective of organisational structure is the ability to define a role as being a part of another role, with the same system applicable to organisational units. This approach allows the model designer to build a model that is simple in its core, but branches out as necessary.

Intangible aspects of an organisation are constrained to the concepts of knowledge artefacts. Individual knowledge artefacts and organisational knowledge artefacts contain knowledge applicable to the elements of a given system, but they discern individual knowledge that is meaningful and important to an individual agent, from organisational knowledge that is applicable to system-level aspects, and is important to any given role.

In the context of MMORPGs, individual knowledge is tied to an individual character, thus describing their attributes such as character traits, skills, history, or inventory; organisational knowledge is used by the given character when they play a certain role, such as rules of conduct in a certain area of the game, available ways of approaching a given mob character, or in-game time.

Strategical aspect of an organisation is realised using the concept of objective along with its available properties and attributes. The most important property of the objective concept, in the context of strategic planning, is `precedentTo` since it defines which objectives precede which other objectives, thus creating a flow of objective concepts which ultimately describe a basis for an action plan. Since lowest-level objectives are achievable by single actions offered by roles defined in the given system, their combination describes which actions are necessary for the fulfilment of their ultimate top goal. Another feature of the Lamrast−+ metamodel should be mentioned here – that of programming code template generator which uses the modelled precedence of objective elements and renders a plan-like code available for use in the modelled system's implementation process afterwards.

Lamrast−+ metamodel contains elements that are necessary for defining actions that organisational units of a system can perform within the system. Such actions stem from the defined roles of the system, since roles here represent grouped norms of the given system. Even though use of actions is recommended, processes can also be defined, as sets of actions i.e. as elements that consist of actions. Using actions alone is recommended since actions are directly used for achieving certain goals. However, it is possible to define an action and its goal, whilst defining it as a part of a process as well.

Individual agents cannot be modelled in detail using this metamodel, yet their presence can. Furthermore, it should be noted that the organisational unit concept of the metamodel is by default used as an agent class when the model is defined, not as a representation of a single individual agent. This view is aligned with the MASs development platform used in this research (SPADE) which allows agents to be defined as classes with their many instances. Yet, an organisational unit concept can be used to represent individual agents, since no formal obligations are set. The metamodel by no means allows the model developer to define implementation-level details of individual agents, since implementation depends heavily on the used programming language and implementation platform.

Organisational dynamics is described in the metamodel using features provided by the tool in which the metamodel is developed. Therefore, organisational dynamics which is realised using graph grammars depends in its implementation on the modelling tool developed along with this metamodel, and is described in Section 2.2.2. Since the model that can be built based on this metamodel in its essence describes a state of a given system, it can be used to describe the system at a single moment in time. Therefore, organisational dynamics is shown using two different instances of the model. Modelling an organisational unit as having the possibility of being a part of another organisational unit shows the intention of developing organisational dynamics during the modelled system's implementation process.

Inter-organisational aspects are present inasmuch as the organisational unit concept can be a representation of either an individual organisational unit, or a group of individuals. No significant difference of these two concepts should be made when the model of a given system is being built, such as roles that can be played, or mutual relationships of organisational units. Therefore, inter-organisational aspects describable using the concepts defined by the metamodel can be modelled.

Some of the mentioned features were already shown on examples from the domain of the *recipeWorld* [43], as well as some other application domains of LSMASs, e.g. MMORPGs [96, 92, 93]. The example of *recipeWorld* is presented in Section 4.1, and The Mana World example is presented in Section 4.2. Both of the example descriptions are used as a medium for highlighting some of the described seven perspectives of organisational modelling of LSMASs [118], an overview of which is shown in Table 2.3. The most complex modelling perspective – context and inter-organisational aspects – is not applicable to these quite simple examples, and is more successfully shown on the third example described in Section 4.3.

Table 2.3: Description of how concepts of the metamodel can be used on two distinct application domains

Perspective	**recipeWorld**	**The Mana World**
organisational structure	The system is described using only individual organisational units, therefore disallowing them to form organisations beside the top-level one represented by the modelled system itself.	Individual organisational unit (a single player character played by an agent) can be a part of an organisational unit – such a relationship represents party or guild membership.
organisational culture	The system defines certain norms some of which are formalised as roles.	Modelled indirectly using the concept of knowledge artefacts – storage of normative elements not included in the definitions of modelled roles.

Continued on next page

Table 2.3 – continued from previous page

Perspective	recipeWorld	The Mana World
strategy	Objectives are described using two complex objectives pertaining to either of the defined roles. Complex objectives are decomposed to atomic objectives achievable by single actions.	Available actions within the system are defined and related to specific roles that can be played by individual agents.
processes	The defined objectives are achievable by various actions that organisational units can perform when playing a role of the modelled system.	Defined actions have their effect on the system environment defined through their connections to the defined objective elements.
individual agents	The system is described using only individual organisational units.	
organisational dynamics	Not applicable.	A relationship exists between an individual organisational unit and a compound organisational unit. A role that can initialise the process of creating compound organisational units is defined.
context and inter-organisational aspects	Not applicable.	Not applicable.

End of Table 2.3

2.2.2 Organisational Dynamics

Organisational dynamics is the concept that involves all the processes that affect organisational features of an organisation, thus introducing change to the observed system. These changes are mostly visible in the organisational structure of a given system, although other features of an organisation can be affected as well. Various elements [118] are deemed needed to tackle the problem of organisational dynamics in LSMASs, since static systems are good enough for implementations featuring individual agents, but are lacking when a multitude of agents is considered, especially when a multitude of agent organisations is considered.

The problem of organisational dynamics is considered in this thesis only from the aspect of organisational structure, thus only in the sense of individual organisational units and them belonging to complex organisational units. Such a problem is described as follows, in terms of graph grammars, temporal and satisfiability logics. Graph grammars are chosen for their applicability to graphs which are the basis of models developed using the Lamrast−+ metamodel, and can be implemented using the customised A Tool for Multi-formalism and Meta-Modelling (AToM^3) modelling tool. Temporal logic is considered a useful addition to graph grammars since dynamic changes in organisational features are happening in time, and are presented herein as events in discrete time. Satisfiability logic is chosen as a tool for describing the environment of a change in organisational features – events before and after the given event.

A solid introduction to graph grammars is provided in [115, 37], with emphasis on active graph grammars in [119]. A short definition of graph grammars is given in [115] using a finite set of productions of graph grammars, whereby a production *is, in general, a triple* (M, D, E) *where* M *and* D *are graphs (the "mother" and "daughter" graph, respectively) and* E *is some embedding mechanism.* A production can be applied to the host graph H when an occurence of M is detected in H. Then, this M is removed from H, and replaced with D or its isomorphic copy, followed by using the embedding mechanism E to finally attach D to the remainder H^- of H. Following [115], there are two types of embedding that can be distinguished: gluing and connecting, based on which two main approaches to graph grammars exist: gluing approach (algebraic), and connecting approach (algorithmic).

The main distinction of the two approaches is in their treatment of nodes and edges of the original (host, H) graph and the additional (daughter, D) graph [115]:

- *In the* gluing *case, certain parts (i.e., nodes and edges) of* D *are identified with certain parts of* H^-.
- *In the* connecting *case, certain new edges are used as bridges that connect* D *to* H^-.

Further theoretical details about graph grammars are provided in Appendix B.2, while this section provides specific details on using graph grammars for organisational dynamics in the context of Lamrast−+ metamodel.

Since graph grammars used alongside the Lamrast−+ metamodel are based on edges and nodes identified using labels, a label alphabet is to be defined in the first place. A

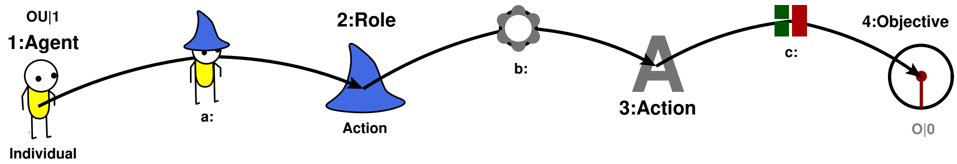


Figure 2.15: An example of an oversimplified model

label alphabet $\mathcal{L} = \langle \mathcal{L}_V, \mathcal{L}_E \rangle$, where $\mathcal{L}_V$ is a set of node labels, and $\mathcal{L}_E$ is a set of edge labels. Elements of both of these sets come from the elements of the metamodel, i.e.

$\mathcal{L}_V = \{$ OrgUnit, Process, Role, Action, Objective, IndividualKnArt, OrganisationalKNArt, $\ldots \}$

$\mathcal{L}_E = \{$ hasAction, hasObjective, playsRole $\}$

A graph built using the Lamrast$-+$ metamodel is therefore a graph over $\mathcal{L}$ is defined as $G = (V_G, E_G, s_G, t_G, l_G, m_G)$, where V_G and E_G are sets of nodes (vertices) and edges respectively, $s_G, t_G : E_G \to V_G$ are source and target functions respectively, and $l_G : V_G \to \mathcal{L}_V$ and $m_G : E_G \to \mathcal{L}_E$ are labelling functions for nodes and edges respectively.

Specifically, for the model in Fig. 2.15, the following is true:

$$V_G = \{1, 2, 3, 4\}, E_G = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\},$$

along with the following:

$$s_G(\mathrm{a}) = (1), s_G(\mathrm{b}) = (2), s_G(\mathrm{c}) = (3);$$

$$t_G(\mathrm{a}) = (2), t_G(\mathrm{b}) = (3), t_G(\mathrm{c}) = (4);$$

whereby labels are distributed as follows:

$$l_G(1) = \mathrm{Agent}, l_G(2) = \mathrm{Role}, l_G(3) = \mathrm{Action}, l_G(4) = \mathrm{Objective},$$

$$m_G(\mathrm{a}) = \varnothing, m_G(\mathrm{b}) = \varnothing, m_G(\mathrm{c}) = \varnothing.$$

Using an analogous approach, any model developed using the Lamrast$-+$ metamodel can be defined in a formal way. The more interesting part are production rules, or simply put, productions. Using the principles of the double pushout (DPO) approach, a production is described by a pair $L \xleftarrow{l} K \xrightarrow{r} R$ of graph homomorphisms from a common interface graph K, where another way [55] of writing the stated is as $p = \langle L \leftarrow K \rightarrow R \rangle$. L is always called a left-hand side, and R is the right-hand side, with K being the interface of p. Further details are specified in Appendix B.2.

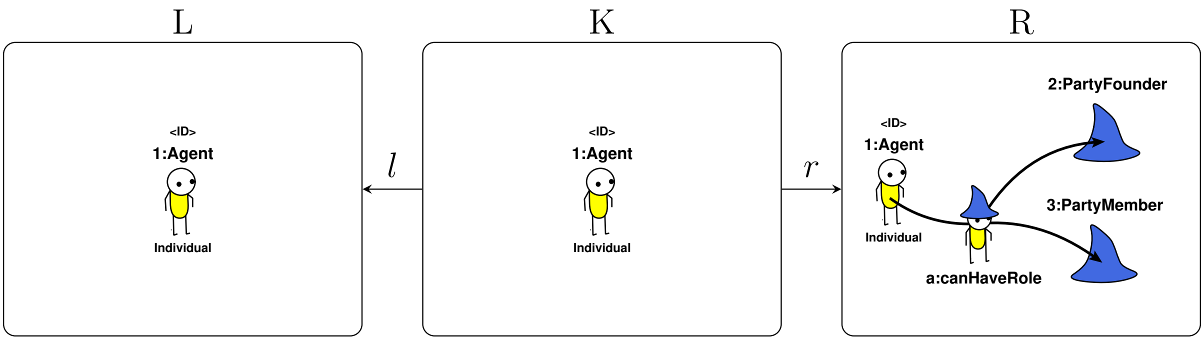
In order to model organisational dynamics, two productions are defined, as shown in Table 2.4.

As mentioned before, creating coalitions or enjoying the privilege of being a part of one

Table 2.4: Production rules

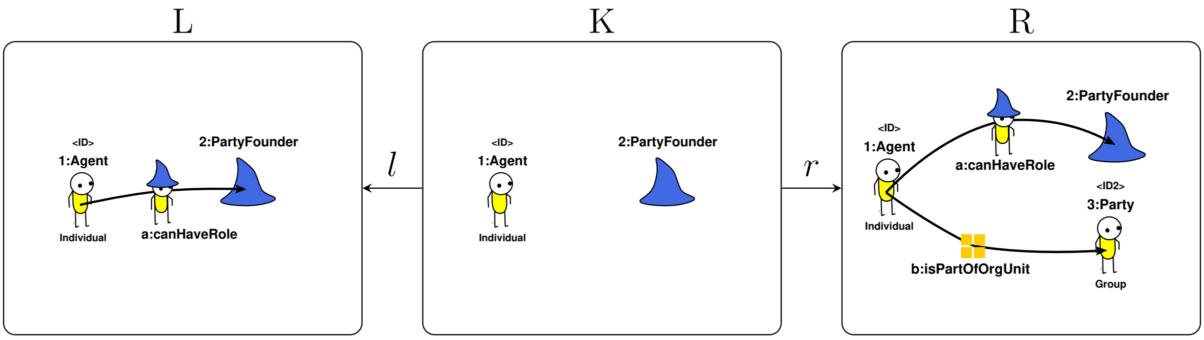
a) Add Roles

Production rule that creates roles for creating and joining groups



b) Enable Grouping

Production rule that creates a higher-level organisational unit



is a process of temporal nature – it happens in time, usually in a set order. So as to show this temporal component of the organisational dynamics in the context of MMORPGs, linear temporal logic [151, 48, 102, 79] is used here. A set of discrete moments T is defined as $T = \{t_1, \ldots, t_a, t_b, \ldots, t_n\}$, where t_a is the moment immediately before the observed event, and t_b is the moment immediately following the event. Based on the language used in [102], referencing [79], several temporal operators are available to be used – those for the future are: N (Next), A (Always), Ev (Eventually), U (Until), and W (Unless or waiting for). In addition to temporal operators for the future, the following temporal operators are used for the past: N^p (Previous), A^p (Has always been), Ev^p (Once), U^p (Since), and W^p (Back to). Where F and G are formulae, so are $N(F)$, $A(F)$, $Ev(F)$ FUG, and FWG, using temporal operators for the future, and $N^p(F)$, $A^p(F)$, $Ev^p(F)$, FU^pG, and FW^pG, using temporal operators for the past.

Furthermore, the temporal context is enriched with operators for expressing agent's knowledge of the system wherein it's located. Hence, the organisational dynamics examples are shown in temporal relation to agent's knowledge of the system. Knowledge operator is defined in [102], referencing [79], as follows. A set of formulae above a set of basic propositions P, and a set of agents A is defined recursively. If every basic proposition from P is a formula, and F and G are formulae, so are $\neg F$, $(F \wedge G)$, $(F \vee G)$, $(F \Rightarrow G)$, and $(F \Leftrightarrow G)$. Finally, if F is a formula, so is $K_i(F)$, $\forall i \in A$ whereby K_i is the modal knowledge operator.

Before organisational dynamics of this metamodel is described here using graph grammars, the context of the example should be set, considering temporal component is rather important when dealing with dynamical processes. Figure 2.16 shows the time-based analysis of a simple MMORPG situation that can be narratively set as follows.

In an MMORPG world, there are two players: Alice and Bob, i.e. $\mathcal{P} = \{\text{Alice}, \text{Bob}\}$. The world is here observed in discrete time periods with $\mathcal{T} = \{0, 1, \ldots, 14, 15\}$. Alice started her life in the in-game world earlier than Bob, at the beginning of the observed time, t_1, or put more precisely, Alice becomes available and present in the system at the transition of time period t_0 to t_1, denoted here, where necessary, as ${}_0t_1$. Bob becomes available later, i.e. at the moment t_4, from its very beginning, therefore from ${}_3t_4$. When the player is not occupied with solving a quest, they are designated as available, therefore `isAvailable(alice)` means that Alice is available (not solving a quest) at the given point in time, shown visually in Fig. 2.16 in horizontal lane `isAvailable(A)`.

Every player has a set of three skills – strength, dexterity and intelligence. Starting value of each of these skills is 0, with the possibility to grow as certain quests are solved by the given player. This growth depends on the defined rewards awarded for successfully solving quests. For example, the `killMaggots` quest rewards the player who finishes it with $(1, 1, 0)$ in skills. The value of skills of a given player is shown in Fig. 2.16, in horizontal lane `skills(A,B,C,D)`, for both Alice and Bob. Skills are therefore noted as

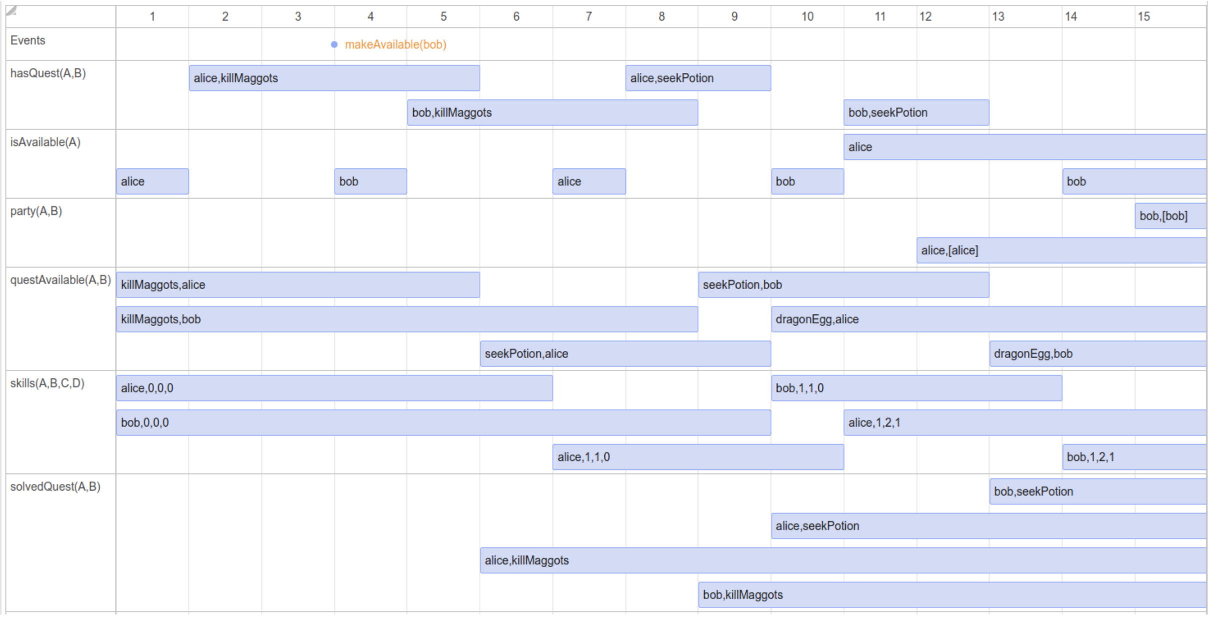


Figure 2.16: Context of the graph grammars example described using LPS, complete code listed in Appendix C.1

`skills(p, s, d, i)`, where $s, d, i \in \mathbb{N}$, and $p \in \mathcal{P}$.

Quests that are available to any of the given players are shown in Fig. 2.16 in horizontal lane `questAvailable(A,B)` denoting which quest is available to which player, e.g. `killMaggots ,alice` reads that the quest `killMaggots` is available (e.g. was unlocked) to the player Alice. Every quest has a set requirements, existing of the minimum value of skills that is necessary for a player to match in order to start solving the given quest. The meaning of a quest being available to a player is that the player can interact with the given quest, but not necessarily that the given player can play the observed quest, i.e. the player does not have to meet the skill criteria of the given quest.

If a player is available and has a quest available that it can start solving (i.e. satisfies its requirements), then they will start the given quest, denoted as `hasQuest(P,Q)` meaning that player P started solving a quest Q. This is shown in Fig. 2.16 in the topmost horizontal lane `hasQuest(A,B)`.

Upon solving a quest, the player who finished it receives the set reward thus advancing through the given game. A line of quests in an MMORPG is defined, and subsequent quests become available to players when they solve their prerequisites.

The situation shown in Fig. 2.16 can therefore be described as follows. The two players, Alice and Bob, start their adventure in an MMORPG world at different times (more precisely, ${}_0t_1$ and ${}_3t_4$ respectively). Both of them can initially start only one quest, labelled `killMaggots`, since it requires no special set of skills. Once Alice solves this first quest, at ${}_5t_6$, its successor is unlocked (quest `seekPotion`), at ${}_5t_6$. Since Alice receives the reward for solving the first quest at ${}_6t_7$, she can start solving the next quest, `seekPotion`.

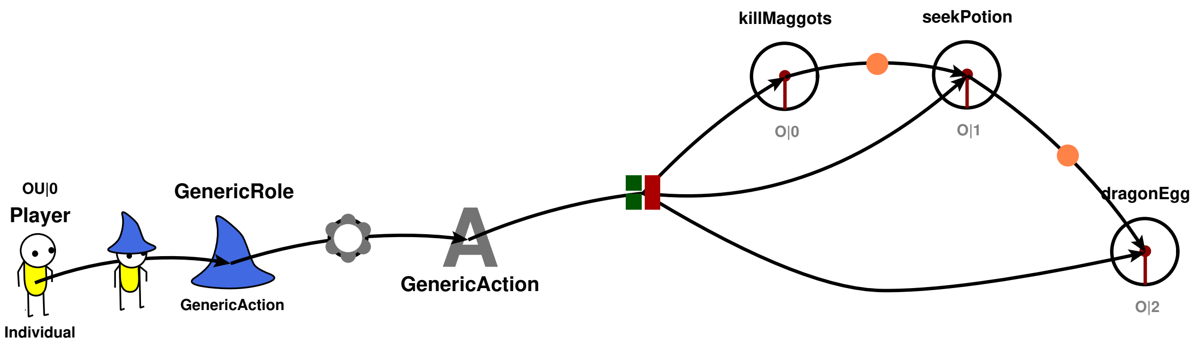


Figure 2.17: Abstracted model representation of the system whose behaviour is shown in Fig. 2.16

Bob is solving his first quest at that time. Once Alice solves her second quest, at ${}_9t_{10}$, the third quest becomes available to her. The reward is not enough for her to bring her skills to the level necessary to start solving the third quest (`dragonEgg`), and there are no other quests available to her, so Alice decides to start looking for help by founding a party at ${}_{11}t_{12}$. Such a party is the basic organisational construct in a MMORPG. At the moment ${}_{14}t_{15}$ Bob acts selfishly and starts his own party for all the same reasons Alice did the same a couple of moments earlier.

In the context where $\mathcal{P}$ is a set of players, and $\mathcal{Q}$ is a set of quests, if the quests a player can play are designated as `canPlayQuest(P,Q)`, where $P \in \mathcal{P}$, and $Q \in \mathcal{Q}$, then individual gameplay for a player P is a valid choice as long as $\exists x : canPlayQuest(P,\ x)$. It is reasonable to expect that at a moment in the future, there will be no quests that a player can play, although a set of quests is available to them, i.e. $Ev(\neg canPlayQuest(P,Q) \wedge questAvailable(Q,P)) : P \in \mathcal{P}, Q \in \mathcal{Q}$. At this point in time, the given player starts playing the role of a *party founder* or a *party leader*, and can create a party. Contrariwise, the given player can assume the role of a *party member*, search for existing parties, and join the one they judge fit. These roles are considered here to be defined by a compound organisational unit by default, yet their creation in the model is subject to graph grammars because organisational units can exist that do not favour grouping of lower-level organisational units.

A simplified model using the Lamrast−+ metamodel that models the interesting parts of the system described here, and shown in Fig. 2.16, is shown in Fig. 2.17. Roles and their actions are not of importance here, and are hence substituted with a generic role and a generic action, both of which should in a real model be expanded into a number of roles and their actions. Furthermore, only the top-level objective is shown, without further deconstruction. Objective sequence is shown though, with `killMaggots` being the first objective (actually representing the concept of a quest in an MMORPG) to be solved, followed by `seekPotion`, and finally `dragonEgg`.

The first graph grammar (shown in Table 2.4) takes place at the moment t_f when the formula $(\neg canPlayQuest(P, Q) \wedge questAvailable(Q, P)) : P \in \mathcal{P}, Q \in \mathcal{Q}$ becomes true, i.e. when there are quests available to a player, but the given player cannot start solving any of those quests since they cannot meet the necessary requirements. Following the rule of graph grammars dual-pushout approach, visualised in Fig. 2.21, and using the graph of Fig. 2.17 as a given graph G, production *Add Roles* from Table 2.4 can be used as shown in Fig. 2.18a. In order to have a clearer situation when working with graph grammars, the following examples use a subgraph of the graph shown in Fig. 2.17, i.e. the graph shown in Fig. 2.19, consisting only of elements representing organisational units, roles, and the relationships between them. Therefore, graph shown in Fig. 2.19 is used in the graph grammars modification processes as the initial given graph G.

Theoretical overview considering productions and pushouts and the generalised process of modifying an initial graph to the resulting graph, is given in Appendix B.2.

The *Add Roles* production (Table 2.4) can be therefore shown as a pushout shown in Fig. 2.18a, using graph in Fig. 2.19 as the initial graph G. The result of applying the stated graph grammar production to the initial graph which consists of an organisational unit that can play a set of roles that are defined by the given organisation the organisational unit is a part of, is the ability of the organisational unit to play a new set of roles consisting of two key roles for modelling the grouping ability of an organisational unit, as shown in Table 2.4: `PartyFounder`, and `PartyMember`. Both of these roles define actions that work with the concept of grouping: `PartyFounder` role enables the organisational unit to create and define new groups of organisational units, while the `PartyMember` role provides actions needed for searching for existing higer-level organisational units, determining how interesting they are to the given lower-level organisational unit, and finally joining them.

Graph H in Fig. 2.18a represents a part of a system that features an organisational unit that can play roles `PartyFounder`, and `PartyLeader`, nonsimultaneously, and is therefore ready for creating, or joining, a higher-level organisational unit. Such a system is used for the example shown in Fig. 2.16 – the featured organisational units (individual agents Alice and Bob) can found parties, or can look for and join existing parties.

The next step is actually forming a party, or in general an organisational unit of a higher level. The production shown in Table 2.4 is the appropriate one for describing this transition – from an organisational unit that can form a higher-level organisational unit, to the one that is a part of a newly formed higher-level organisational unit. The initial graph G is given in Fig. 2.20, as an isolated part of the graph representing the whole system, just as was case above, when the *Add Roles* production was considered. Production *Enable Grouping* in Table 2.4 is a graph grammars approach of what can be described verbally as an organisational unit founding a higher-level organisational unit, where it is a leader and a founding member.

Clearly and graphically put, double pushout of the *Enable Grouping* production is shown in Fig. 2.18b, where the final graph H features an organisational unit that can be a part of another organisational unit – the one it just created.

In the context of the LPS example visualised in Fig. 2.16, this second graph grammar production (*Enable Grouping*) takes place at the moment t_s, which immediately follows t_f when organisational units are introduced to the appropriate roles. In particular, agent Alice started the first higher-level organisation, of the observed example, at t_{12}.

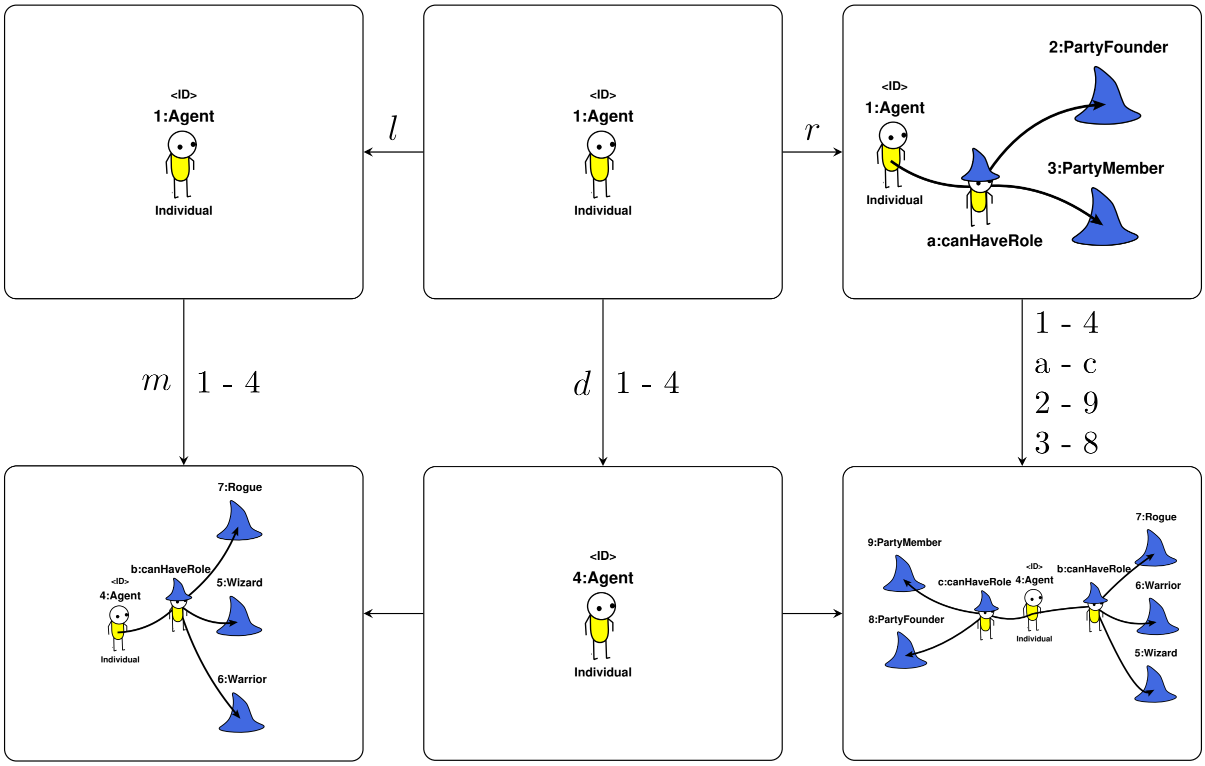
All of the graph grammars derivations are implemented in the chosen metamodelling tool, and can be used when working with the metamodel. These can be used in the process of modelling an LSMAS using the Lamrast−+ metamodel. Howbeit, the metamodel is intended to be used as a static representation of the modelled system, wherefore all of the derivations described here have no real impact on the generated code template for the modelled system, as they are of no use to the system once the implementation phase is realised – detailed behaviour of agents is not a concern of this research as of yet. An interesting future research may be a real-time model following the changes in a developed model's system in runtime.

Considering the nature of the metamodel's representation of an individual organisational unit, i.e. considering the fact that the organisational unit element of the metamodel does not represent an individual of a class, but rather a class of organisational units (e.g. the class of Player agents, and not the individual player Alice), the model's graph representation is not very rich in expressions, yet it shows many aspects of organisational dynamics. In the case of a model like in Fig. 2.22, where two organisational unit elements are present, along with a role element representing a set of custom roles, two roles used for the process of organisational dynamics (*PartyFounder*, and *PartyMember*), their respective actions (not presented in the referenced model), the verbal decription and interpretation is as follows:

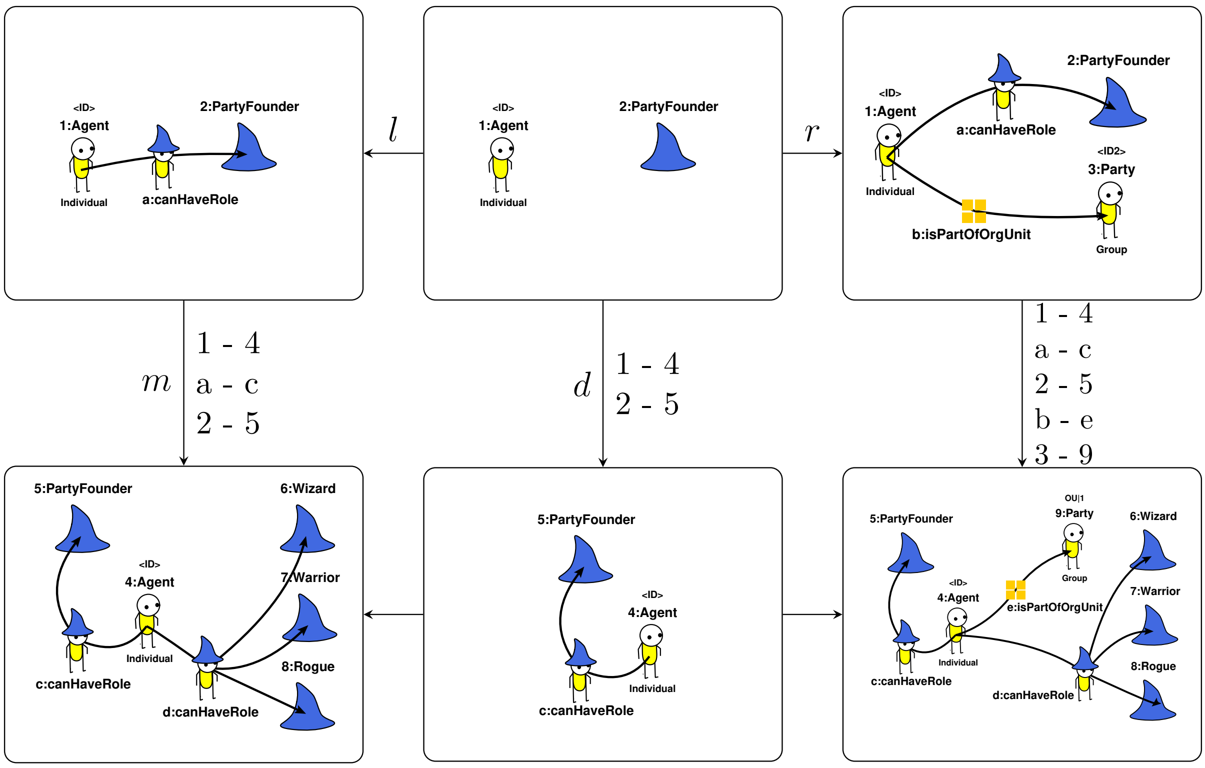
> A lower-level organisational unit can play any one of the roles from the custom set of roles at any given moment. At the same moment, the organisational unit can play any one of the following two roles as well: *PartyFounder* and *PartyMember*. Using the actions provided by the *PartyFounder* role, the organisational unit can establish a higher-level organisational unit. On the other hand, if a higher-level organisational unit is present, the organisational unit can, using the actions provided to them by playing the *PartyMember* role, look for, assess, and join a higher-level organisational unit.

It is possible therefore that the higher-level organisational unit is only an abstract concept consisting of real or artificial agents, or an implemented agent – such a decision is in the hands of the modelled system's developers, and is not of concern at the modelling stage of a system's development. An example of the higher-level organisational unit as an abstract concept is the Fellowship of the Ring[3] (from the legendarium of J.R.R. Tolkien),

[3]For more information, visit `http://lotr.wikia.com/wiki/Fellowship_of_the_Ring`



(a) Double pushout of production *Add Roles*



(b) Double pushout of production *Enable Grouping*

Figure 2.18: Double pushouts of the defined productions

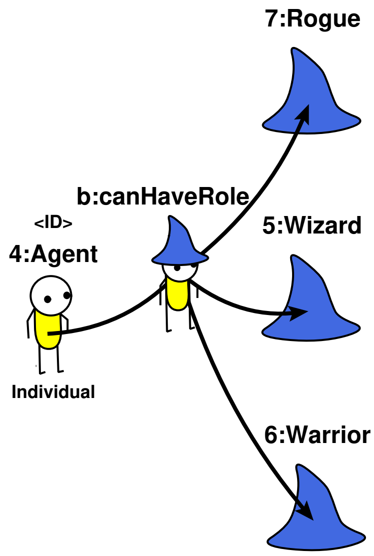


Figure 2.19: The initial graph G suitable for *Add Roles* production

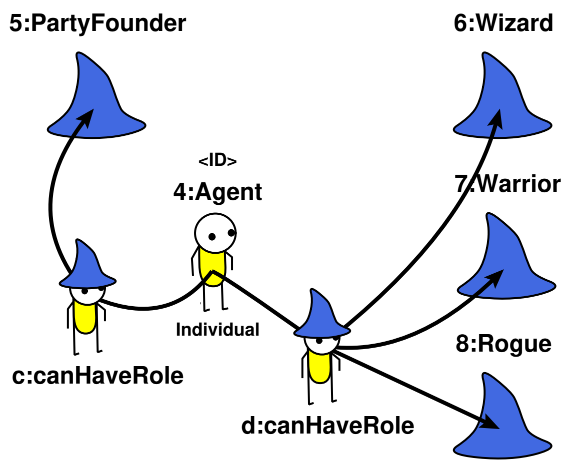


Figure 2.20: The initial graph G suitable for *Enable Grouping* production

$$
\begin{tikzcd}
L \arrow[d] \arrow[dr, phantom, "(1)"] & K \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(2)"] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}
$$

Figure 2.21: DPO approach structure, a direct derivation, according to [37]

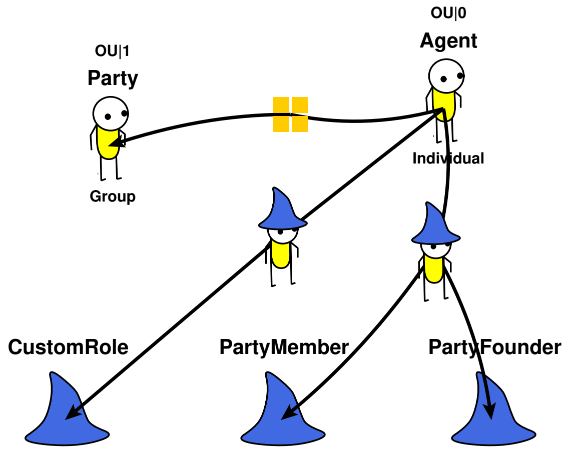


Figure 2.22: Model with necessary elements for dynamic organisational structure

a brotherhood of members of the various Free Peoples of Middle-Earth, which consists of nine agents, where the Fellowship is only a name for the defined group of agents. A similar situation is with the Avengers[4] from Marvel's universe. The higher-level organisational unit in this context provides no new features, other than the combined power of individual agents it consists of, and their cooperative effort towards fulfilling a common goal. On the other hand, a Megazord[5] – a combination of five Dinozords – from the Mighty Morphin Power Rangers live-action television and movie series, can be considered, in the context of higher-level organisational units, as a new agent, since it is not an abstract concept, but a combination of lower-level agents, thus forming a new agent with features that are not merely the combination of those of the included lower-level agents, but surpass them.

[4] For more information, visit `http://marvelcinematicuniverse.wikia.com/wiki/Avengers`
[5] For more information, visit `http://powerrangers.wikia.com/wiki/Megazord`

Chapter 3

Practical Contribution

Even though many models applicable to the domain of multiagent systems (MASs) have been defined, and published in many a research, some of them described in subsection 1.4.3 and [3, 92], only a few of them have had their practical application developed, i.e. their development somehow ended with theoretical definitions and guidelines.

The goal of the development process of Lamrast−+ metamodel is not to leave it on theoretical level, thus providing only a sense of scientific contribution, but to move further on to developing a metamodelling tool that uses the concepts defined by the metamodel, which can be used for modelling complex large-scale multiagent systems (LSMASs). Furthermore, apart from modelling systems comprising agents, the metamodelling tool provides the used with the feature that allows them to generate an implementation template for the modelled system. Therefore this chapter provides the description of the practical contribution of this research.

This chapter describes the developed metamodelling tool (defined as a modification of an existing tool used for metamodelling), its application guidelines, some of the features and challenges. The content of Sections 3.1 and 3.2 describes the tool and how Lamrast−+ metamodel is implemented, while the feature of generating implementation template is covered in Section 3.3.2.

The tool can be found online as a publicly available open source project at GitHub, `https://github.com/Balannen/LSMASOMM`.

3.1 Metamodelling Tool

Apart from defining the sole metamodel, a complete metamodelling process can go further, towards defining various constraints introduced by the metamodel yet possibly not visible in the graphical representation of it, in order to provide a wholesome metamodelling approach. Building blocks of such a *modeling method*, as referenced to by [66], include the modelling language, the modelling procedure, and the mechanisms and algorithms. The modeling language is described [66] as a set of modelling constructs along with their grammar and semantics – syntax (grammar) in the context of defining possible fundamental modelling constructs, and semantics as unambiguous meaning of the constructs of the language. Modelling procedure is the part that *defines the steps that must be taken by modelers towards their goal.* [66] Amongst the steps it defines are the precedence guidelines on what should be the order of creating certain types of models so as to have an ultimately valid model. The block dealing with mechanisms and algorithms covers various forms of functionality in the context of processing models and their content for a number of purposes such as visualisation, transformation, simulation, etc.).

Building further on the described *modeling method*, it is stated that a modelling tool, especially a domain-specific one, should include:

> "(a) model-driven functionality that is relevant with respect to the modeling requirements; (b) guidelines and constraints for modeling scenarios with respect to different modeling goals and related functionality." — Karagiannis et al. [66]

In the context of creating a modelling tool that introduces practical application to a defined metamodel, a *model of a formalism should contain enough information to permit the automatic generation of a tool to check and build models subject to the described formalism syntax.* [33]

Two metamodelling tools, i.e. tools that allow the user both to define a metamodel, and use the defined metamodel to develop a model representation of an observed system, that are observed as a part of this research are the ADOxx[1] and A Tool for Multi-formalism and Meta-Modelling (AToM^3)[2]. Some fundamental differences between them are: the wealth of features, the ease and practicality of adding new or external features, technical details, licences used, and more (some of these is presented in Table 3.1). The most important similarity is that both these tools provide their users with the ability to define a metamodel, and to use the defined metamodel when creating a model.

Granted, other tools that utilise the metamodelling process exist, such as those from the Eclipse community[3], yet only ADOxx and AToM^3 are considered here since the author has most experience with them. Furthermore, both of them fulfill the above stated features of a modelling tool and a metamodel from [66, 33].

For further discussion provided in this document AToM^3 [109, 32, 33] is used, mainly because it is completely developed using Python programming language, and it is entirely open source, fostering its customisation based on the needs of Lamrast$-+$ metamodel, and the process of metamodelling in the context of additional features and constraints. Furthermore, being developed in Python, it can easily be connected to Smart Python Agent Development Environment (SPADE), which is the MASs development platform

[1]For more information, visit `https://www.adoxx.org/live/home`
[2]For more information, visit `http://atom3.cs.mcgill.ca`
[3]For more information, visit `https://www.eclipse.org`

Table 3.1: Selected similarities and differences of ADOxx and AToM^3

	ADOxx	**AToM^3**
Platform dependency	restricted to Windows	can be installed and run on both Windows and Linux
Availability	free	free
Source code	closed	open source
Metamodelling	graphic interface	graphic interface
Custom code in metamodel	using AdoScript, a proprietary language	using standard Python
Customisation opportunities	the tool is available as is	the tool can be customised as needed

of choice in this research. Finally, Python community is rich in various modules and extensions, thus allowing for successfully effective constraint development and setup of a dynamic tool component featured as actions, which can be customised.

The reason SPADE, as a particular MASs development platform, was chosen for its implementation in Python which makes the agents developed using it widely applicable since they can be enriched using some of the numerous community-developed modules, and being the first such piece of software ever to use a particular popular communication protocol (XMPP) [93]. Furthermore, it is completely open source[4], developed in academia, and open to community upgrades.

A survey of 24 agent platforms compared against a set of criteria was conducted by Kravari and Bassiliades [70]. Since SPADE is not featured in this survey, it is evaluated here according to the criteria used in the referenced survey. An overview of a different set of agent programming platforms and languages is provided in [132, chapter 5].

> "Platform properties refer to the primary concepts of the platform, describing its basic characteristics that are necessary for a potential user/developer in order to understand the scope and the domain of the platform. Usability refers to the suitability of the platform for the construction of agent applications. Operating ability refers to all these aspects that are taken into account during execution. In other words, operating ability indicates the quality of the platform. Pragmatics refers to external factors that are neither related to the construction nor to the operation of the platform. More specific, pragmatics indicates whether the platform can be used in practice or not. Finally, security management refers to security issues, indicating if the platform is considered safe or not." — Kravari and Bassiliades [70]

[4]For more information, visit `https://github.com/javipalanca/spade`

Table 3.2: Evaluation criteria used by Kravari and Bassiliades [70]

Platform properties	Usability	Operating ability	Pragmatics	Security management
Developer / Organisation	Simplicity	Performance	Installation	End-to-end security
Primary domain	Learnability	Stability	User support	Fairness
Latest release	Scalability	Robustness	Popularity	Platform security
License	Standard compatibilities	Programming languages	Technological maturity	
Open source	Communication	Operating systems	Cost	

Based on the set of criteria in Table 3.2, SPADE is evaluated as shown in Table 3.3, according to data available as of September 2018. Detailed description of each criteria is available in [70].

It should be noted here that there are only two agent platforms in the referenced survey [70] that use Python, yet both are based on Java, and require Java Virtual Machine to be run, on any platform. Furthermore, none of the surveyed agent platforms offers compatibility with the XMPP/Jabber technology. SPADE, however, is developed entirely in Python, therefore allowing developers to naturally use all the available Python modules and expansions.

3.2 Metamodel Implementation

The working metamodel that can be used with AToM^3 metamodelling tool, as shown in Fig. 2.14 on Page 63, was developed using the formalism creation feature of AToM^3.

3.2.1 Basis for the metamodel

Lamrast−+ metamodel was therefore defined as a new model, using concepts from the AToM^3 predefined class diagram consisting of elements shown in Fig. 3.1 (classes, associations, and inheritance). Class element is used for various classes of Lamrast−+ metamodel, associations are used for various defined properties of the classes, and inheritance is used rarely, but a use case exists within Lamrast−+ metamodel.

Every element of a model defined in AToM^3 can be defined using several key attributes, as shown in Fig. 3.2:

name the name of the element, defining how the element is referenced, formatted following set rules for naming Python variables[5], where personal preference is the so-called CamelCase;

Graphical_Appearance defines how the concept will be represented graphically when AToM^3 is used;

[5]For more information, visit `https://www.python.org/dev/peps/pep-0008/#naming-conventions`

Table 3.3: Evaluation of SPADE according to criteria used by Kravari and Bassiliades [70]

Platform properties	
Developer / Organisation	Development project led by Javi Palanca and Gustavo Aranda, with significant contributions by Markus Schatten, Juan Angel Garcia-Pardo, and Santiago M. Mola Velasco
Primary domain	General purpose multiagent systems (including large-scale distributed systems)
Latest release	Latest GitHub commit dated 7 September 2018
License	Creative Commons Attribution License
Open source	Yes
Usability	
Simplicity	Simple, administrative-only web interface available
Learnability	Easy
Scalability	High
Standard compatibilities	Communication protocols based on XML (e.g. FIPA-ACL), FIPA-SL, RDF
Communication	XMPP, P2P, HTTP, SIMBA
Operating ability	
Performance	High
Stability	High
Robustness	Good
Programming languages	Python, plus RDF, Prolog, XML
Operating systems	Linux, Windows
Pragmatics	
Installation	Command line
User support	Average (docs, email)
Popularity	Low
Technological maturity	Stable release, Development status (Active)
Cost	Free
Security management	
End-to-end security	N/A
Fairness	N/A
Platform security	N/A

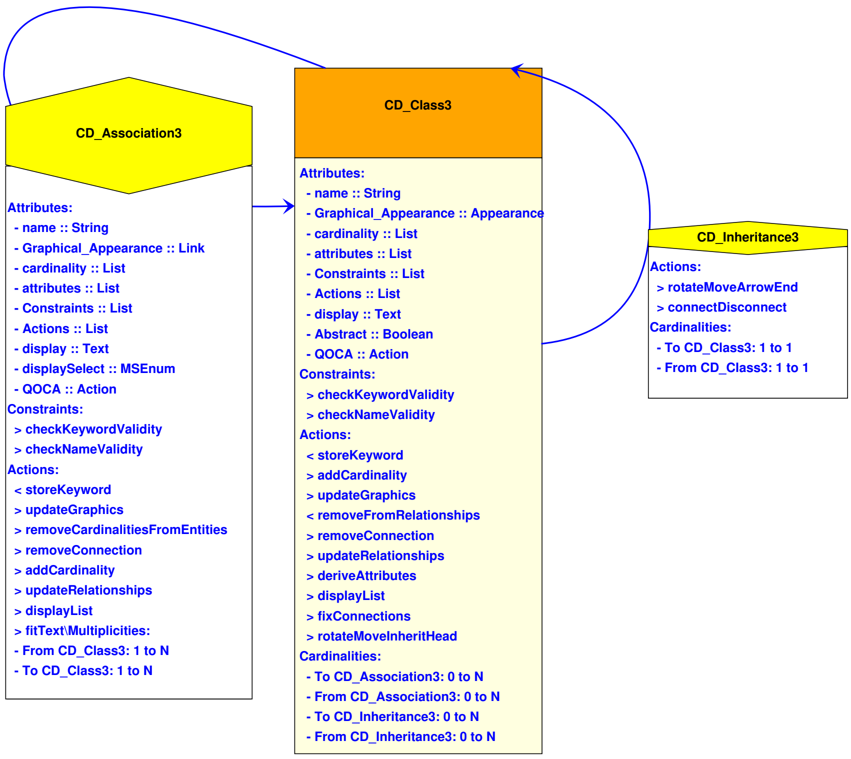


Figure 3.1: The elements of AToM^3 predefined class diagram metamodel

cardinality a set of associations that are connected to the specific class individual, and what their relationship is with the given class individual (e.g. a destination, or a source, and minimum and maximum cardinality);

attributes a set of attributes that will be available for use if and when the element is going to be used as an element of a metamodel, along with their core properties (e.g. name, type, initial value, if the attribute is a key attribute uniquely identifying the individual, and if the value of the attribute can be directly modified from the individual edditing window);

Constraints a set of customised Python code snippets that can be introduced as implementation of various constraints that act as either preconditions or postconditions, with defined names and proposed triggers – should a constrant return anything but a *True* value, the action which was constrained will not finish and will be recalled;

Actions much like the Constraints attribute, the Actions attribute is a set of actions that are realised as Python code snippets defined by their name, their nature (pre- or postaction), and their triggers;

display defines what textual content is displayed in the visual representation of the model (like the one in Fig. 2.14).

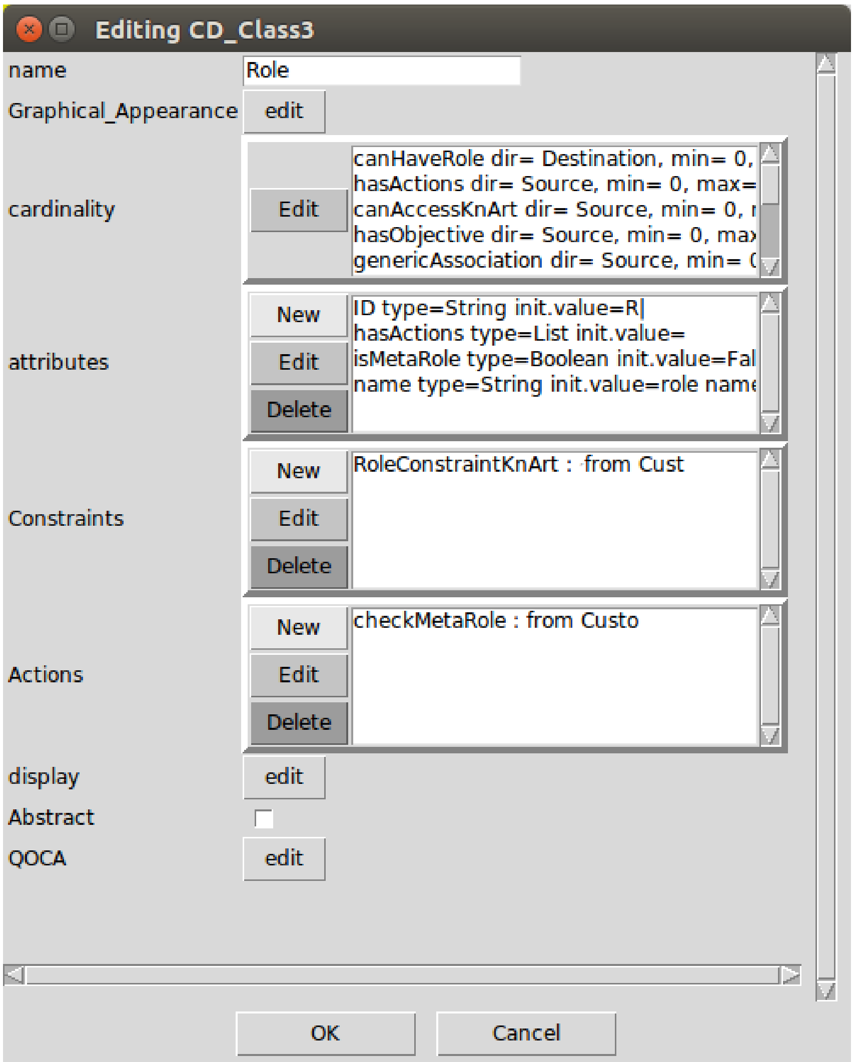


Figure 3.2: Editing attributes of a class diagram class individual

Abstract this boolean attribute defines whether the class is going to be an abstract class or not, and will therefore restrict individual creation, or allow it, respectively;

QOCA again a piece of Python code, this is a specific QOCA type of constraint that can be defined.

The above list of attributes that can be defined for an instance of a class concept, are a very good example of the relationship of a model and it's metamodel. Those attributes are defined as element attributes (of the element named *class*) in the model describing a class diagram, shown in Fig. 3.1. Since the class diagram model is used as a metamodel for Lamrast−+ metamodel, elements of Lamrast−+ metamodel, which are instances of the class element of the class diagram metamodel, can be defined using the defined attributes. Similarly, element attributes defined in Lamrast−+ metamodel are used for further defining their instances in a model that describes a multiagent system, or its large-scale version, based on Lamrast−+ metamodel. In other words, elements of a model based on Lamrast−+ metamodel have an interface such as that in Fig. 3.2, but with attributes defined in metamodel, as shown in Fig. 2.14 on Page 63.

Not all of the attributes shown in Fig. 3.2 have to be defined manually, such as *cardinality* which is defined based on the connections, i.e. properties, defined in the graphic layout of the model. What can be expressed here is the details about cardinality of a connection.

`Graphical_Appearance` attribute contains graphical representation of the element in the model view. Graphical appearance is based on Tkinter[6], with possible addition of GIF[7] elements. Graphical appearance of all the elements in Lamrast−+ metamodel are defined using Tkinter only, for the sake of visualisation quality, scalability, and usability. In addition to defining static graphical elements, AToM^3 allows the developer to add some dynamic parts to an element's graphical appearance, which change based on the value of attributes or are changed by element constraints or actions.

`Constraints` and `Actions` attributes are the most similar to amongst all the attributes of a class element of the class diagram metamodel. Both of these are realised as a piece of Python code that is run either as a *pre* or *post* event, and are triggered by one of the following actions of the developer:

Edit is triggered when the element's attributes or other properties are edited;

Save is the action of saving the model being developed;

Create triggers when the instance of a concept is created;

Connect is run when two elements are connected to each other, whereof at least one is the element which has the action or constraint set to run at this particular trigger;

Delete triggers when the element is deleted;

Disconnect is run when two elements are disconnected from each other, whereof at least one is the element which has the action or constraint set to run at this particular trigger;

Transform is triggered when the element's graphical appearance is transformed;

Select triggers when an element is selected;

Drag triggers when the element's graphical appearance is picked by the model's developer to be moved across the canvas in AToM^3;

Drop triggers when the element's graphical appearance is dropped by the model's developer after being moved across the canvas in AToM^3;

[6]Tkinter is Python's de-facto standard GUI (Graphical User Interface) package; for more information, visit `https://wiki.python.org/moin/TkInter`

[7]Graphics Interchange Format; for more information, visit `https://en.wikipedia.org/wiki/GIF`

Move is the action of moving the element's graphical appearance across the canvas in AToM^3.

Most of the above triggers are element-based, with the only exception being the `Save` trigger, which is run when the whole model is saved.

The difference between the `Constraints` attribute and the `Actions` attribute is designated by the treatment of their code – while actions are there simply to perform some action, a constraint has the power to cancel an action that is being performed as it's being triggered. In other words, a constraint code is run before (*pre*condition) or after (*post*condition) an action is performed, with the power to cancel the given action, or reverse it, based on the outcome of the constraint code. An action, on the other hand, is a piece of code that is performed before (*pre*action) or after (*post*action) an action is performed, without necessarily directly affecting the action itself, rather a graphical appearance of the element, the value of its attributes, or anything else. Furthermore, since all the elements of a model are connected, actions and constraints can modify, or be based on, values of other connected elements.

3.2.2 Defining the Metamodel

As was mentioned before, the elements (concepts) of Lamrast−+ metamodel are defined as individuals of class diagram metamodel's `Class` and `Association` concepts, with seldom use of `Inheritance` concept.

The instances of `Class` concept are:

- `Role`,
- `OrgUnit`,
- `Action`,
- `OrganisationalKnArt`,
- `IndividualKnArt`,
- `KnowledgeArtifacts`,
- `Objective`,
- `Process`,
- `Strategy`.

The instances of `Association` concept are:

- `isPartOfOrgUnit`,
- `answersToOrgUnit`,
- `canHaveRole`,
- `canAccessKnArt`,
- `isPartOfRole`,
- `answersToRole`,
- `genericAssociation`,
- `hasActions`,
- `canStartProcess`,
- `isPartOfProcess`,
- `hasObjective`,
- `isPartOfObjecctive`,
- `precedentTo`.

The concept of inheritance is used to designate that both `Objective` and `Process` concepts inherit some attributes from `Strategy` concept, and that `OrganisationalKnArt` and

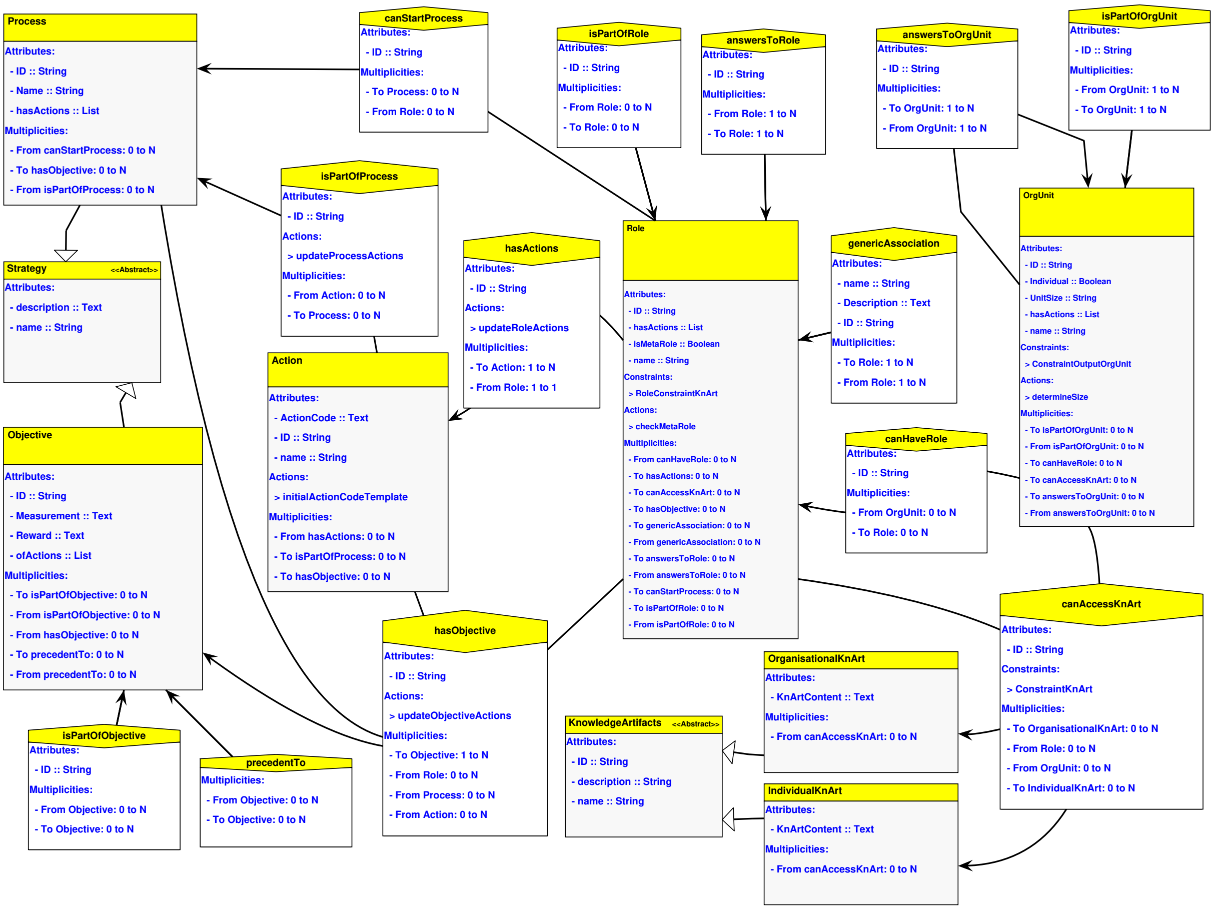


Figure 2.14: Repeated visual representation of Lamrast−+ metamodel from Page 63

`IndividualKnArt` inherit some attributes from `KnowledgeArtifacts` concept. Other than attribute inheritance, such a relationship has no further benefits for the implementation of Lamrast−+ metamodel.

The named concepts, and how they are connected, is shown visually in Fig. 2.14, repeated here for the sake of accessibility.

Since most of the attributes are self-explanatory, with some of the concepts used in the metamodel described in Appendix A.1, only an overview of the chosen metamodel concepts is given hereafter.

Role The role represents a set of normative constraints that are not given literally and explicitly, but are modelled using a grouping concept of a Role. A role allows organisational units to play accompanying actions, thus enabling them to affect the system they're a part of. A role can be a part of another role, using a specialised form of an inheritance property, similar to the *is a* property of Resource Description Framework (RDF). The role concept can access `OrganisationalKnArt` concepts only.

OrgUnit The organisational unit concept is defined using the same presumptions as explained in other places in this thesis, namely the recursive approach. An organisational unit element has an attribute which defines it as an individual agent or a group of agents. Every organisational unit individual can access an unlimited number of roles, can be designated as being a part of another organisational unit, and can access only an individual knowledge artefact concept instance.

Action An action is the basic form of how an organisational unit playing a role can affect its environment, i.e. the system wherein it is located. Every action individual is associated with its respective role concept individual, and its respective objective concept individual – an action can be enacted by an organisational unit concept individual playing a respective role concept individual, with the goal of achieving a respective objective concept individual.

Process A process is a set of actions that are grouped for a reason and can be performed in a sequence. As such, a process represents a form of a strategy, since execution of the actions of a process is an attempt of achieving a set objective.

Objective An objective is a state of the system that an organisational unit is looking forward to achieving. Objectives are designated as complex or elementary, based on them being composed of other lower-level objectives, or being on the lowest level of objective decomposition, respectively. An elementary objective can be achieved directly by an action concept individual, while a complex objective is achieved with regard to its sub-objectives' status. An objective concept individual can thus be a part of another objective concept individual, and a precedence association can be defined, as a sort of a strategic directive for an organisational unit.

canHaveRoles This association concept connects organisational unit individuals to role individuals, thus representing which roles can be played by which organisational units. Each instance of this association represents a logical disjunction in the context of an organisational unit having a set of roles offered for playing at a given moment.

The peculiar nature of the organisational unit concept (OrgUnit in Lamrast−+ metamodel) is its behaviour in the context of it being used in a model. Namely, the organisational unit concept from the metamodel is instantiated as an organisational unit individual in a model. The meaning of that individual depends on the will of the developer – it can represent an entity that can directly be implemented and instantiated, and that acts on its own, or it can represent a class of entities that will be instances or individuals of that particular class of entities. In other words, if used in the context of SPADE, the organisational unit concept in the model based on Lamrast−+ metamodel will most likely represent a class of agents, since SPADE allows the developer to define a class of agent, that can have individual agents instantiated at runtime. Such an approach is used in examples in [92]. A different approach can take an organisational unit element in a model as a representation of a single agent of the modelled system. Both approaches are permitted as per the metamodel's design.

Apart from the basic act of defining metamodel concepts using the attributes provided by the class diagram metamodel, an important role is played by the additional programming code developed for the purposes of constraints and actions of the metamodel concepts, but for other features of the metamodelling tool, such as generating application template (described in more detail in Section 3.3.2), and support for multimodel modelling. This additional custom code is presented in Section 3.3.

3.3 Custom Code

Some of the features of the final metamodelling tool were developed using custom Python code. Even though some of the features realised using custom code are basic, it may have been easier to implement them using customised code, rather than fine tuning all the features of AToM^3. Customised code is therefore used to various ends, from simply modifying graphical appearance of model elements, to constraint implementation, to development of support for multimodel modelling, and generating application templates based on the systems modelled. The file containing most of the customised code (excluding that which is scattered throughout AToM^3 implementation, is available on GitHub[8].

One of the basic functions developed for the purpose of code used in constraints and actions is `NodeOutputsInputs` which is used by other functions to receive a set of nodes or a number of nodes that are neighbours of the given node – on the source or the destination end of an association. This particular function was implemented with the goal of reducing code redundancy, since a similar feature was sought after in many other customised functions. The function therefore returns to its caller either a set of nodes, or simply their number, of either nodes on the other end of in- or out-connections, sorted by their respective concepts, as per request of the caller function.

The use of this function is exemplified further using another function, `OrgUnitDetermineSize`, which is used as an action of `OrgUnit` concept, since an organisational unit is designated as `individual` if there are no lower-level organisational unit concept individuals connected to the given one, and `group` when there is at least one lower-level organisational unit concept individual connected to the given one, i.e. if the given organisational unit is a higher-level organisational unit concept individual relative to another organisational unit concept individual. The code for this function is given in Listing 3.1. The above mentioned `NodeOutputsInputs` is called in line 2 of Listing 3.1, whereby only a count of nodes by their class is wanted, for all the in-connections, i.e. all the nodes on the source sides of in-connections of the given organisational unit concept individual's node. The function

[8]For more information, visit `https://github.com/Balannen/LSMASOMM`

```python
def OrgUnitDetermineSize(self):
    eIns = NodeOutputsInputs(self, 'in', 'count')

    if 'isPartOfOrgUnit' in eIns:
        return 'Group'
    elif 'isPartOfOrgUnit' not in eIns:
        return 'Individual'

    return
```

Listing 3.1: Implementation details of function *OrgUnitDetermineSize*

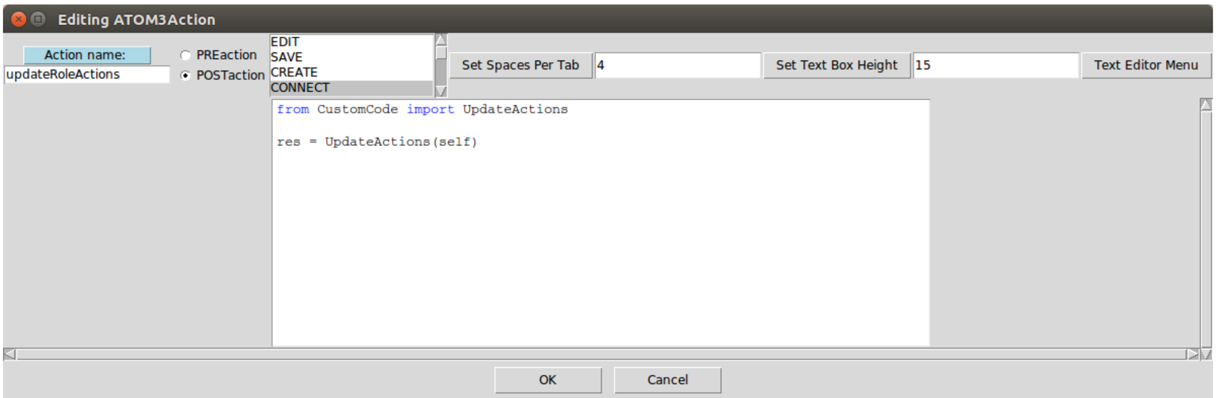


Figure 3.3: Editing *updateRoleActions* action of *hasAction* concept

gives a certain value in return, based on its environment. The return value is further analysed and acted upon in AToM^3.

Another good example of customised code defined for *Actions* attribute of a role concept is `UpdateActions` function, which populates the list of actions of a role concept individual based on the action concept individuals connected to it. Thus the attribute of a role concept individual is always updated if there is a change on the graphical level.

This function is implemented as shown in Listing 3.2. Here the `NodeOutputsInputs` is used again, to retrieve the nodes that are on the either side of an in- or out-connection. Graphical appearance modification in implemented in line 15, while the list of actions is prepared as shown in line 9, where data must be prepared as a predefined *ATOM3String* data type to be an eligible element for a list of values.

This particular **action** was added to the `hasActions` concept in the metamodel, as shown in Fig. 3.3. The action is, as visible in Fig. 3.3, defined as a *post*action triggered by a connect or disconnect (not visible in the figure) event. When triggered, the defined piece of code is run, i.e. the `UpdateActions` function from `CustomCode` file is called. Since all the modifications are performed as a part of the called function, nothing additional has to be defined in the action code itself.

Another example of an **action** that shows how customised code communicates with

```python
def UpdateActions(self):
    eOuts = NodeOutputsInputs(self, 'out', 'nodes')
    eIns = NodeOutputsInputs(self, 'in', 'nodes')

    actions = []

    if 'Action' in eOuts:
        for a in eOuts['Action']:
            actions.append(
                prepareAttributeValue('ATOM3String', a.name.getValue()))

        if 'Role' in eIns:
            for r in eIns['Role']:
                for a in actions:
                    r.hasActions.newItem(a)
                r.graphObject_.ModifyAttribute('hasActions', r.
                    hasActions.toString())
            return 1

    return 0
```

Listing 3.2: Implementation details of *UpdateActions* function

various elements and features of a model based on Lamrast−+ metamodel, is `ActionCodeTemplate` function, which is called as a part of `initialActionCodeTemplate` action of the action concept. The action is set up as shown in Fig. 3.4 – as a postaction triggered by a create event, thus being run when an action concept individual is created. The action code calls `ActionCodeTemplate` function from the file of customised code, listed in Listing 3.3.

The customised code for `ActionCodeTemplate` is a bit more complex, as it works directly with attributes of the whole model (defined on the metamodel level as well), as opposed to working only with the attributes of the given individual. Line 2 in Listing 3.3 is looking for the model being developed by the name of its metamodel. Value of its `agentImplementation` attribute is returned in line 3, and is used in lines 5 through 13 to determine what should be the returned template. At the moment, the only agent implementation feature provided by the modelling tool is that of SPADE. The selected code template is thereafter formatted as an AToM^3 text type data, and is returned as such to the action code of the action concept individual.

The action code then modifies the value of *ActionCode* attribute of the given action concept individual, thus giving the model developer a code template to work with, based on the designated agent platform. What the generated action code template looks like as an attribute value when an action concept individual is edited, is shown in Fig. 3.5.

A good example of a **constraint** implementation is given as a constraint of a *canAccessKnArt* concept. The constraint artfully named `ConstraintKnArt` is defined as a *post*condition triggered by a connect event, as shown in Fig. 3.6. The constraint code, shown in the figure, but listed here in Listing 3.4, is used to interpret the return value of the called

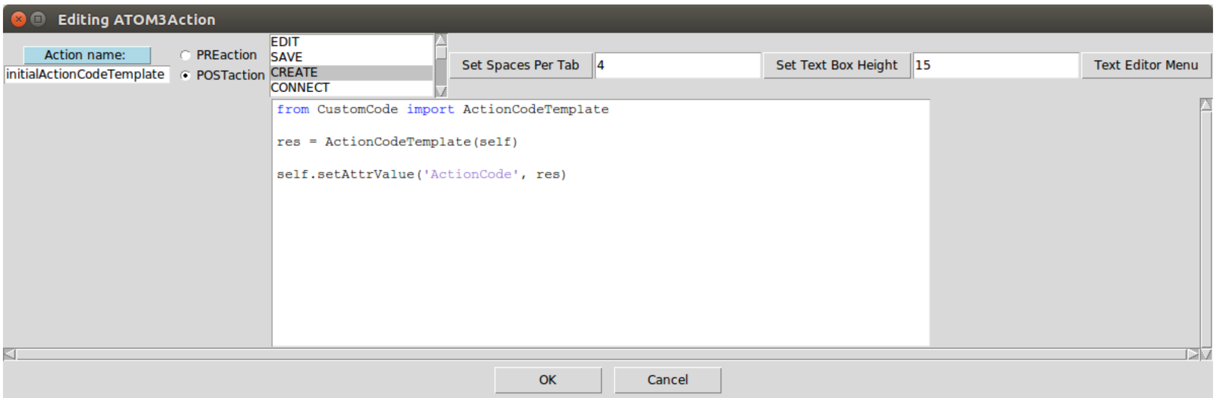


Figure 3.4: Editing *initialActionCodeTemplate* action of *Action* concept

```
def ActionCodeTemplate(self):
    Root = self.parent.ASGroot.getASGbyName('LSMASOMM_META')
    t, s = Root.agentImplementation.getValue()

    if t[s] == 'SPADE':
        codeString = u'''#action code template
class BehaviourNamePlaceholder(spade.Behaviour.OneShotBehaviour):
    """Behaviour available to agents."""
    def _process(self):
        pass
'''
    else:
        codeString = ''

    codeTemplate = prepareAttributeValue('ATOM3Text', codeString)

    return codeTemplate
```

Listing 3.3: Implementation details of *ActionCodeTemplate* function

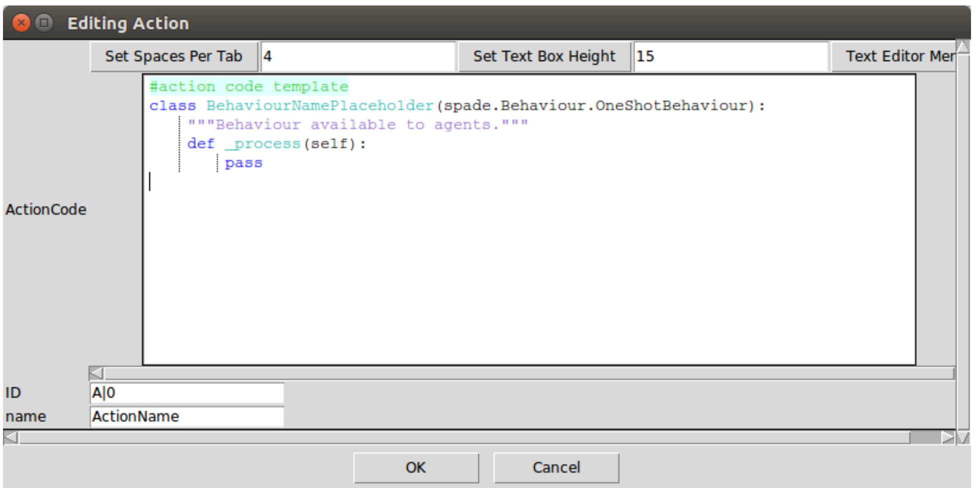


Figure 3.5: Editing an *Action* individual

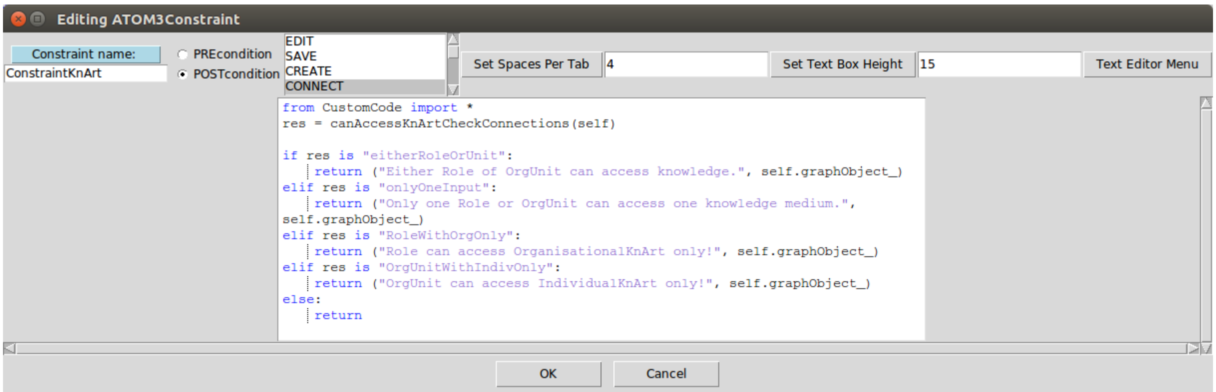


Figure 3.6: Editing *ConstraintKnArt* constraint of *canAccessKnArt* concept

`canAccessKnArtCheckConnections` function in `CustomCode` file. This function is shown here in Listing 3.5.

The original function code (Listing 3.5) checks the number of assorted nodes at the ends of incoming and outgoing connections (lines 2 and 6), and returns an according return value. For example, if a role concept individual is on the incoming connections side (relative to the given *canAccessKnArt* concept individual), and an individual knowledge artefact concept individual is on the far end of the outgoing connections side (again, relative to the same concept individual), defined by line 7, the function returns a specific keyword `RoleWithOrgOnly` with the meaning that roles can only be connected to organisational knowledge artefacts (line 8). If no constraints are validated, the function returns no specific value (line 11).

Back in the constraint details (Listing 3.4), the behaviour of the constraint is ruled by the function's return value – if anything is returned, the constraint is fired up because a specific value is returned and a graphical representation of a model element is associated with the constraint (the returned value is used as a warning message). If no specific value is returned by the associated function, the action that triggered the constraint is left as is. Otherwise, the connect action which triggered the constraint, is undid (since it was already done as the constraint is defined as a postcondition). If the associated function returned keyword `RoleWithOrgOnly` (line 6), the constraint is invalidated and the appropriate warning message is shown to the metamodel user (line 7).

This is a good example to illustrate the difference between a pre- and postcondition type of a constraint (and similar approach is used for actions as well). If this particular constraint was run before the connections were established, i.e. as a *pre*condition, the associated functions would not be able to assert the situation according to the set constraint. Therefore, the action which triggered the constraint would go unnoticed until the next such action was performed – only then would the results of the last connection action be visible. On the other hand, as a *post*condition, the constraint is run after the whole

```
from CustomCode import *
res = canAccessKnArtCheckConnections(self)

if res is "eitherRoleOrUnit":
    return ("Either Role of OrgUnit can access knowledge.", self.graphObject_)
elif res is "RoleWithOrgOnly":
    return ("Role can access OrganisationalKnArt only!", self.graphObject_)
elif res is "OrgUnitWithIndivOnly":
    return ("OrgUnit can access IndividualKnArt only!", self.graphObject_)
else:
    return
```

Listing 3.4: Implementation details of *ConstraintKnArt* constraint

```
def canAccessKnArtCheckConnections(self):
    eIns = NodeOutputsInputs(self, 'in', 'count')
    if 'Role' in eIns and 'OrgUnit' in eIns:
        return 'eitherRoleOrUnit'

    eOuts = NodeOutputsInputs(self, 'out', 'count')
    if 'Role' in eIns and 'IndividualKnArt' in eOuts:
        return 'RoleWithOrgOnly'
    if 'OrgUnit' in eIns and 'OrganisationalKnArt' in eOuts:
        return 'OrgUnitWithIndivOnly'
    return
```

Listing 3.5: Implementation details of *canAccessKnArtCheckConnections* function action is performed, and can therefore assess the situation correctly. If the performed action is against the constraint, the results of the action, since it is performed already, are annihilated, and the pre-action state of the model is reinstated.

3.3.1 Multimodel Modelling

One of the earliest problems that were encountered whilst the metamodelling tool was being developed was that AToM^3 canvas would get very crowded and hardly legible even when only a simple model was being constructed, based on Lamrast−+ metamodel. The limited, but great in terms of available space in an AToM^3 canvas, number of model elements hindered legibility and usability of the model, since the graphical representation is, after all, meant for human agents. This problem coupled perfectly with the idea of modelling organisational units recursively, and made it necessary and opportune to modify the modelling tool in a way that would support an approach to modelling large-scale models through many smaller linked models – a multimodel modelling approach.

Since the number of concepts necessary for successful description of a small snippet of a massively multi-player online role-playing game (MMORPG) world comprising only one quest, such as the one described in [124, 125, 96, 93, 92], is quite great for the space available in AToM^3 modelling canvas, the idea of defining a model using a number of models was captivating. Furthermore, it was recognised later that the multimodel modelling approach is beneficial even for filtering and clustering wanted or temporarily needed elements, drawing only the necessary out of the whole set of available elements, i.e. elements that were defined earlier.

Further argument in support of the multimodel modelling approach is derived from the research in knowledge management, where one of the tendencies is to work towards knowledge reuse. Building upon the lines of knowledge reuse, it is possible to reuse any of the previously defined model elements, as long as they come from the active metamodel, namely from Lamrast−+ metamodel. Some further constraints apply, but the general idea is achieved.

The multimodel modelling is implemented using a database running in the background – a ZODB[9] database instance written in a file on the client's computer. A separate file is created for every model name. Every model database contains all the concepts defined by the model developer.

The **saving** side of using ZODB is straightforward, inasmuch as the objects are simply to be defined, and are ready for storing data. Storing all the relevant data about all the relevant elements defined in a model based on Lamrast−+ metamodel is handled using customised code in *CustomCode* file. The action of saving model elements is triggered using the *Save All* button in AToM^3 interface when Lamrast−+ metamodel is being

[9] A Python object-oriented database; for more information, visit `http://www.zodb.org/en/latest/` `http://www.zodb.org/en/latest/`

used. Therefore, the *SaveAll* function is run on the model level, as opposed to being run at the model element (concept individual) level, as was the case with the functions described above. The saving process is implemented mainly using *Save All* function, listed in Listing 3.6.

Firstly, the name of the model is gathered from the `name` attribute of the model (line 5), and a database file is created or opened (lines 6-7) using the name specified. Since all the nodes (model elements) have to be saved, it is useful to utilise the list of nodes grouped by node types (concept classes) that is automatically being constructed by AToM^3 – `Root .listNodes`. The list of types used in the model (the concepts of the individuals used in the model) is the set of key of the Python dictionary of all the elements of the given model – `Root.listNodes.keys()`. Lines 10-15 check if a type is already present in the given database file, meaning that it can be used further. I case it is not, the given type is added to the database. Such a logic was designed since it makes it easier to access all the saved nodes when they are saved in a structured Python-like dictionary where they are grouped by types. Furthermore, it makes the loading and implementation template generating processes easier. If no node of the given type has yet been saved (i.e. the type does not exist in the root of the database file), it is added therein, as a persistent object of ZODB (line 14). When the type root is found (line 11) or created (lines 14-15), iteration through all the model nodes of the given type can start, and they can be saved using `SaveNode` function. If the node was saved already (recognised by its ID attribute), an extra argument is sent to `SaveNode` function (line 19).

When all the nodes are saved, a knowledge base (KB) entry is saved as well, in a Prolog-like format describing all the Action-Objective (lines 33-36), Role-Action (lines 38-41), and OrgUnit-Role (lines 43-47) pairs. Therefore, if *Wizard* role defines *CastSpellFireball* action, the associated KB entry would be `('Wizard', 'hasAction', 'CastSpellFireball')`. The values used in KB entries are taken from the model (e.g. OrgUnit-Role pairs are gathered by observing all the *canHaveRole* individuals, and their in- and outconnections) or the individual nodes (Role-Action and Action-Objective pairs are populated by reading their respective node attributes containing role actions, or action objectives respectively, which are then parsed as individual pair values).

Upon introducing a change to the database file, no changes are saved immediately, but a sum of changes can be saved and thus committed to the database file using the `transaction.commit()` function call, as seen in lines 23 and 50. The changes are therefore saved in two batches – the first one saving node modifications and additions, and the second one saving KB modifications.

The second part of saving node data is implemented using `SaveNode` function listed in Listing 3.7. The function is called from `SaveAll` function, and is tasked with saving all the relevant data of a specific single node (concept individual) in the model. The function works along two similar paths depending on whether the database entry should

```
def SaveAll(self):
    global DBname
    Root = self.ASGroot.getASGbyName('LSMASOMM_META')

    DBname = Root.name.getValue()
    db = openDB(DBname)
    conn = db.open()

    for nodeType in Root.listNodes.keys():
        try:
            dbRoot = conn.root()[nodeType]
        except Exception as e:
            print e
            conn.root()[nodeType] = PersistentMapping()
            dbRoot = conn.root()[nodeType]

        for node in Root.listNodes[nodeType]:
            if node.ID.getValue() in dbRoot.keys():
                SaveNode(node, conn, True)
            else:
                SaveNode(node, conn)

    transaction.commit()

    if 'KB' not in conn.root():
        KB = {
            'ActionGoal': {},
            'RoleAction': {},
            'UnitRole': {}}
    else:
        KB = conn.root()['KB']

    for goal in conn.root()['Objective'].values():
        for a in goal.attrs[5].split('\n'):
            if a:  # to avoid empty strings
                KB['ActionGoal'][(a, 'canReachGoal', goal.attrs[goal.
                    realOrder.index('name')])] = True

    for role in conn.root()['Role'].values():
        for a in role.attrs[1].split('\n'):
            if a:  # to avoid empty strings
                KB['RoleAction'][(role.attrs[role.realOrder.index('name'
                    )], 'hasAction', a)] = True

    for link in conn.root()['canHaveRole'].values():
        if 'OrgUnit' in link.in_connections_ and 'Role' in link.
            out_connections_:
            for o in link.in_connections_['OrgUnit']:
                for r in link.out_connections_['Role']:
                    KB['UnitRole'][(o, 'canHaveRole', r)] = True

    conn.root()['KB'] = KB
    transaction.commit()
    db.close()
```

Listing 3.6: Implementation details of *SaveAll* function

```
1  def SaveNode(node, conn, update=False):
2      if update:
3          DBnode = conn.root()[node.__class__.__name__][node.ID.getValue()
               ]
4          DBnode.updateAttributes(
5              node.getStringValue(),
6              node.copyCoreAttributes()[2:4])
7
8      else:
9          DBnode = savedNode(node.copyCoreAttributes())
10         DBnode.saveAttributes(
11             node.realOrder,
12             node.getStringValue())
13
14         conn.root()[node.getClass()].update(
15             {DBnode.ID: DBnode})
```

Listing 3.7: Implementation details of *SaveNode* function

be created and added or simply modified, as described above. In case the node already exists, it is found (line 3) and its `updateAttributes` method is called with two arguments containing all the attribute values in string format, and select core attributes using the `copyCoreAttributes` customised function defined as a method of a node concept. Moreover, some of the core attributes do not change over time, wherefore only the select core attributes are needed (line 6).

If the node is not yet present in the database file, it is instantiated from the object defined for use with ZODB (line 9), and its `saveAttributes` method is used with the appropriate arguments containing the list of node attributes, and their values (lines 10-12). The node is eventually saved in the database file under its type, as a new Python dictionary entity with the key value of its ID attribute.

The class definition developed for saving node objects in ZODB database file is given in full in Appendix C.2. A piece of code is listed in Listing 3.8, for explanation purposes. When the node object (in the database context), is initialised as described above in Listing 3.7, with an argument containing all the node attributes of a given node. When initialised, the database node instance saves those values (lines 3-14 in Listing 3.8). Furthermore, the `saveAttributes` method of the database node class is used for saving values of all the customised attributes (those defined by Lamrast−+ metamodel), as shown in lines 16-18.

The final element of the model saving process is performed continually, triggered whenever two model elements are connected to each other in the given model, implemented as a constraint of the model (defined on the metamodel level as a constraint of the model rather than that of a concept). The function called directly by the constraint is listed in Listing 3.9. The function opens the database file based on the name of the

```
class savedNode(persistent.Persistent):

    def __init__(self, coreAttrs):
        self.graphClass_ = coreAttrs[0]
        self.isClass = coreAttrs[1]
        self.in_connections_ = coreAttrs[2]
        self.out_connections_ = coreAttrs[3]
        self.containerFrame = coreAttrs[4]
        self.keyword_ = coreAttrs[5]
        self.editGGLabel = coreAttrs[6]
        self.GGset2Any = coreAttrs[7]
        self.GGLabel = coreAttrs[8]
        self.objectNumber = coreAttrs[10]
        self.ID = coreAttrs[11]

    def saveAttributes(self, order, attrValues):
        self.realOrder = order
        self.attrs = attrValues
```

Listing 3.8: Excerpt from *CustomCodeDB* shown in full in Appendix C.2

model (as is always the case in customised code), and iterates through all the present nodes, type by type. If the given node is not present in the database file, it is created and stored regularly. Every node's in- and outconnections are scanned for connected nodes, and if the connected node's type is not present in the node's in- or outconnections sets, it is added. Otherwise, if the connected node is not in its respective set, the initial node is set for an update of its in- or outconnections sets. This approach is implemented using lines 16-24 for inconnections, and lines 25-33 for outconnections.

Now that data is stored in a database file, the **loading** part has to be implemented. The process of loading nodes onto AToM^3 canvas is conditioned by a number of factors: the name of the model that is edited is the name of the database file being sought after, just as it was the name of the database file used for saving data; the class of the node that is to be implemented has to be chosen, with the option of selecting the desired node the be loaded opening only after the node type (class) is chosen; the nodes can be loaded one by one or in a set of same-type nodes.

One of the functions necessary for successful implementation of the loading part is concerned with preparing AToM^3 data types, since most of the data in ZODB database files was saved as strings, and AToM^3 nodes demand somewhat customised data types. Once this side-function was implemented, element loading can be successfully implemented.

The node loading feature is implemented using native AToM^3 functions for creating model elements. Once the new node (model element) of the same type as the loaded node is created, all the core and additional attributes are copied from the stored to the created node. Furthermore, since connections are stored as well, the model is scanned for all the nodes designated as those that the stored node was connected to, and if any exist,

```
def addConnectionToDB(self):
    global DBname
    if os.path.isfile("./DB/{}.fs".format(self.name.getValue())):

        try:
            db = openDB(DBname)
            conn = db.open()
        except Exception:
            print "Called from another function (probably when loading concepts)"
            return

        for nodeType in self.listNodes.keys():
            try:
                for node in self.listNodes[nodeType]:
                    if node.ID.getValue() in conn.root()[nodeType].keys():
                        if len(node.in_connections_):
                            inNode = node.in_connections_[-1]
                            DBnode = conn.root()[nodeType][node.ID.getValue()]
                            if inNode.__class__.__name__ not in DBnode.in_connections_:
                                DBnode.in_connections_[inNode.__class__.__name__] = []
                            if inNode.ID.getValue() not in DBnode.in_connections_[inNode.__class__.__name__]:
                                SaveNode(node, conn, True)
                                DBnode.in_connections_._p_changed = 1
                                transaction.commit()
                        if len(node.out_connections_):
                            outNode = node.out_connections_[-1]
                            DBnode = conn.root()[nodeType][node.ID.getValue()]
                            if outNode.__class__.__name__ not in DBnode.out_connections_:
                                DBnode.out_connections_[outNode.__class__.__name__] = []
                            if outNode.ID.getValue() not in DBnode.out_connections_[outNode.__class__.__name__]:
                                SaveNode(node, conn, True)
                                DBnode.out_connections_._p_changed = 1
                                transaction.commit()
                    else:
                        SaveNode(node, conn)
                        transaction.commit()
            except Exception:
                pass

        db.close()
```

Listing 3.9: Implementation details of *addConnectionToDB* function the connection is established again. Loading is therefore implemented in the manner of creating new elements that get the values of their attributes filled in automatically, based on the saved node which is being loaded.

Thus implemented saving and loading of concept individuals, i.e. model elements, makes it possible for the model developer to model the wanted system using several models which focus on varying aspects of the same system, while building a single model nonetheless. This wholesome model stored in a ZODB database file is used as input for the application template generating feature described hereinafter.

3.3.2 Application Template Generator

The final aspect of customised code of the metamodelling tool is the implementation part of the feature of the metamodelling tool using Lamrast−+ metamodel that allows the model developer to generate application template based on the defined model. This feature of this research is the most valuable in the context of practical contribution, as it brings direct benefit to LSMASs' developers.

The application template generating feature uses the metamodel details saved in the accompanying ZODB database file, and generates key implementation parts of the modelled system. A couple of features of the modelled system are covered by the generated template:

- key definitions of modelled organisational units;
- basic code of the modelled actions;
- knowledge base containing OrgUnit-Role, Role-Action, and Action-Objective pairs, defined as knowledge of organisational units, thus simulating organisation-wide knowledge of organisational norms;

The process of application template generation is started by the model developer using the appropriate button in AToM^3 model based on Lamrast−+ metamodel. Such an action simply runs the `generateNodeCode` function of the file with the customised code. The definite result of the whole process is created in cooperation of this code external to model elements, and that of `generateCodeSPADE` method of nodes (model elements) saved in the associated ZODB database file. The complete implementation of `generateNodeCode` function is listed in Listing 3.10.

Analogous to the functions observed above, the associated ZODB database file is to be opened first, and a connection established (lines 3-5 in Listing 3.10). Technicalities are dealt with next, with all generated code being stored in the *Code* folder which is first checked if it exists (lines 7-8).

Actions defined in the model are all stored in a single file, `RoleBehaviours.py`. Action codes are written according to how they are defined in the model, i.e. in action individuals, and their respective `ActionCode` attributes. All action implementation code is used in sequence, by action individual, and written into the same file, `RoleBehaviours.py` (lines 16-17).

Afterwards, organisational units are implemented using `generateCodeSPADE` method of the customised ZODB database object class. The function call is given an argument containing the modelled knowledge base, since the knowledge base is expected to be hardcoded into the organisational unit, for it to be able to use this knowledge from the beginning. Certainly, the final decision whether the knowledge stays with the organisational unit after the full process of development is entirely upon the system's developer. The implementation side of the application template generating feature creates a new file for the respective organisational unit individual, where it is implemented using node attributes (e.g. `name` and `hasActions`), and the provided knowledge base. The mentioned `hasActions` attribute of an organisational unit individual is not to be confused with the same-named one which is a part of every role individual. `hasActions` attribute of an organisational unit individual defines names of actions that are inherently a part of an organisational unit, and that can be performed regardless of the role played by the given organisational unit. One such key action is `changeRole` which enables the organisational unit to change the role it plays. Furthermore, such an action can be performed even when, for example in the beginning, when the system is first launched, the given organisational unit individual has no other options. This set can be further expanded to, e.g. actions that choose another objective for the organisational unit to pursue, or similar. The nature of use of these two similar but different attributes is upon the system or model developer as well.

Finally, a file combining all the generated files is created, where all the organisational unit individuals are ready to be run, and all the actions are imported and ready to be performed by organisational units, along with the details about all the organisational units and their knowledge bases. The application template thus generated is therefore a multi-file implementation from the start.

```python
def generateNodeCode(self):
    global DBname
    Root = self.ASGroot.getASGbyName('LSMASOMM_META')
    db = openDB(DBname)
    conn = db.open()

    if not os.path.isdir("./Code"):
        os.mkdir("./Code")

    filename = './Code/RoleBehaviours.py'
    if os.path.isfile(filename):
        os.rename(filename, '{}.old'.format(filename))

    file = open(filename, 'w')

    for k,v in conn.root()['Action'].items():
        file.write("\n{}".format(v.attrs[0]))
    file.close()

    agents = []

    KB = conn.root()['KB']['RoleProcessGoal'] + conn.root()['KB']['
       RoleActions']

    for k, v in conn.root()['OrgUnit'].items():
        agents.append(v.generateCodeSPADE(KB))

    db.close()

    filename = './Code/TheSystem.py'

    if os.path.isfile(filename):
        os.rename(filename, '{}.old'.format(filename))

    file = open(filename, 'w')
    file.write("import spade\nfrom RoleBehaviours import *\n")
    for agT in agents:
        file.write("from {} import *\n".format(agT))

    file.write('\nif __name__ == "__main__":\n')

    for x in range(0, len(agents)):
        file.write("""
    agent{0} = {1}("{1}{0}@127.0.0.1", "secret")
    agent{0}.start()
""".format(x, agents[x]))

    file.close()
```

Listing 3.10: Implementation details of *generateNodeCode* function

Chapter 4

Examples

The following examples serve the function of Lamrast−+ metamodel evaluation in three contexts related to the concept of LSMASs. Such an evaluation serves the purpose of arguing in favour of the metamodel's *meta* prefix and its applicability on a scale larger than that of the domain of MMORPGs.

All the three examples described hereinafter have their context defined first, and the example described in further detail if necessary, followed by a defined model of the system or its selected part, with the generated application template at the end. Thus every example is presented through the three important steps: the observed source system, the model, and the system that is ready to be implemented.

4.1 recipeWorld

The concept of the recipeWorld is described in [93], with the idea of SPADE implementation of the included concepts referenced in [103], both based on the original paper of the recipeWorld [43].

Described shortly, the recipeWorld is an agent-based model that *simulates the emergence of a network out of a decentralised autonomous interaction.* [43] The combination of agent-based modelling and network analysis, as provided by the recipeWorld model, is deemed beneficial in the context of raised potential of complexity-based policies. The key elements of the recipeWorld are recipes, orders, and agents. Recipes are a list of prerequisites for achieving a certain goal, usually perceived as steps that can vary in number. The aforementioned goals are named orders, as they represent concretisation in the form of objects containing technical information and the necessary data that defines order instances. Agents are problem-solving cores that can provide some services

The model of the described domain can be represented using the Lamrast−+ metamodel as shown in Fig. 4.1. In the context presented in Section 2.2.1.5, the model of the recipeWorld can be described as follows.

The system is described using only individual organisational units, therefore disallowing them to form organisations beside the top-level one represented by the modelled system itself. This organisation defines certain norms, some of which are formalised as roles available in the modelled system (`Order` and `Factory`). Objectives are described using only two top-level objectives pertaining to either a factory or an order. These top-level objectives are decomposed to objectives that can be achieved by single actions. Objective decomposition is separately shown in Fig. 4.2, where their proposed order is designated as well. These defined objectives are achievable by various actions that organisational units can perform when playing a role of the modelled system. Roles and their respective actions are, separated from the rest of the system's model, shown in Fig. 4.3.

This simple-to-understand example is a good starting-point when description of the Lamrast−+ metamodels is being provided.

The model representing recipeWorld, as shown in Fig. 4.1, was developed using the metamodelling tool described in this thesis. Various elements are defined in more or less detail using the available attributes defined at the metamodel level. Editing those values is similar to editing metamodel concepts described in Section 3.2. Details of action `SearchForFactories` is shown in Fig. 4.4, where the associated action implementation code is shown as well. The implementation code defined here can be used in the application template generating feature afterwards.

The referenced model is available on GitHub repository of the research of this thesis[1].

Upon running the application template generation, three files are created or updated if they exist already: one with the initial core code for the modelled organisational unit, one with all the available actions and their respective implementation code (where applicable), and one combining both of these files with the basic SPADE system.

Knowledge base of the modelled organisational unit consists of related organisational units, roles, actions, and objectives, and is listed in Listing 4.1, as a part of a SPADE agent's `_setup` method, which is used for adding behaviours (actions) to agents as well, such as `ChangeRole` action, as shown in line 3, and more. The knowledge base is to be interpreted using the following template: `property('from', 'to')`. Thus, `hasAction('Factory', 'Produce')` is interpreted to mean that `Factory` role defines `Produce` action.

The generated code is not enough for the modelled system to be run though. Nor is that the intention of the model, and it being modelled using the supplied metamodelling tool. Implementation details, necessary for the system to be run, are to be supplied and taken care of by the modelled system's developer.

It should be noted here that modelling is not uniform, i.e. models depend on the needs and perspectives of model developers. An example observation based on Fig. 4.1 is that actions of `Order` role may have been grouped so that three actions (`WaitForFactoryAnswer`, `CheckFactoryAvailability`, and `SearchForFactories`) are represented by a single action that

[1]For more information visit `https://github.com/Balannen/LSMASOMM`

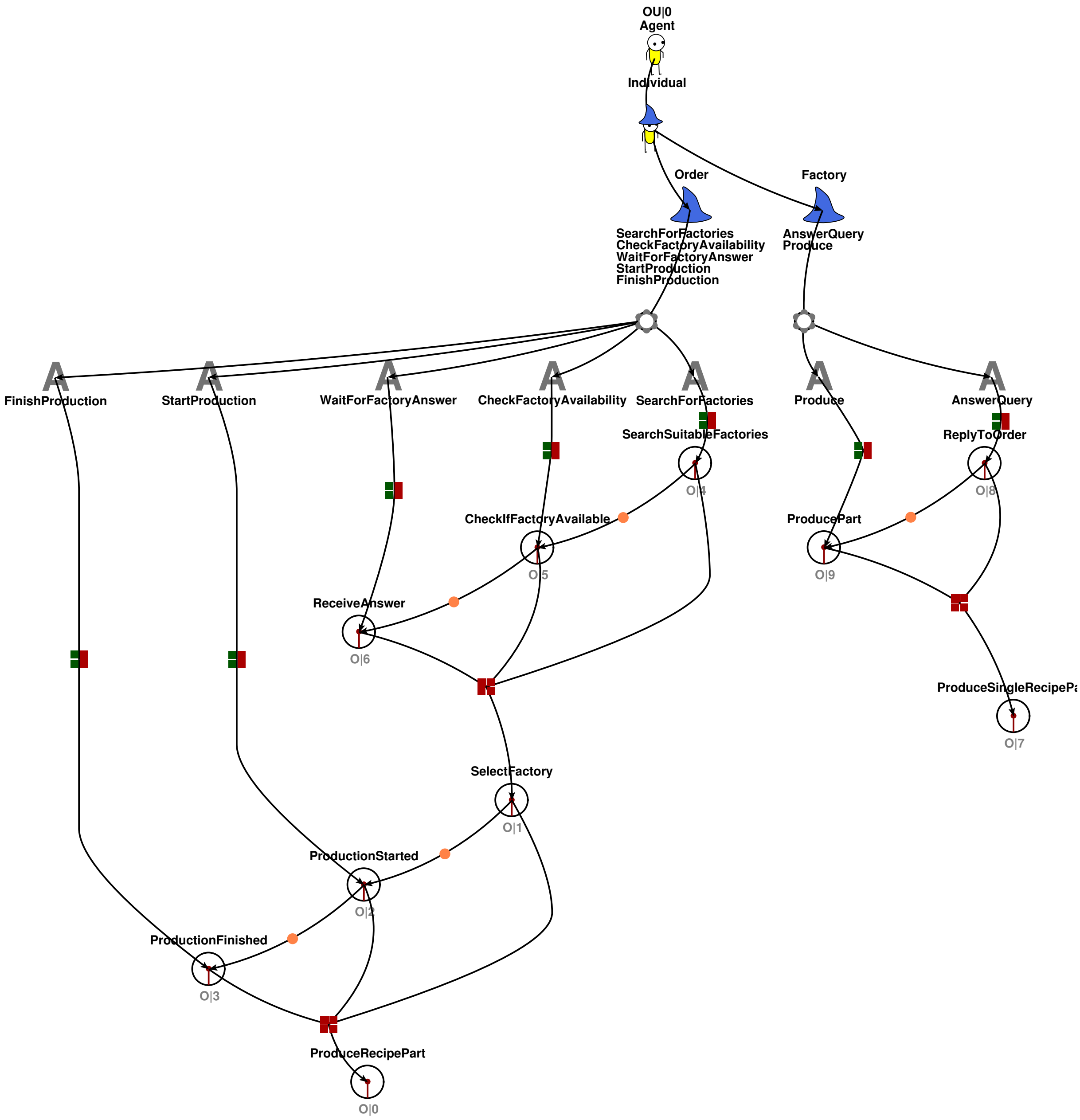


Figure 4.1: The model of the *recipeWorld*

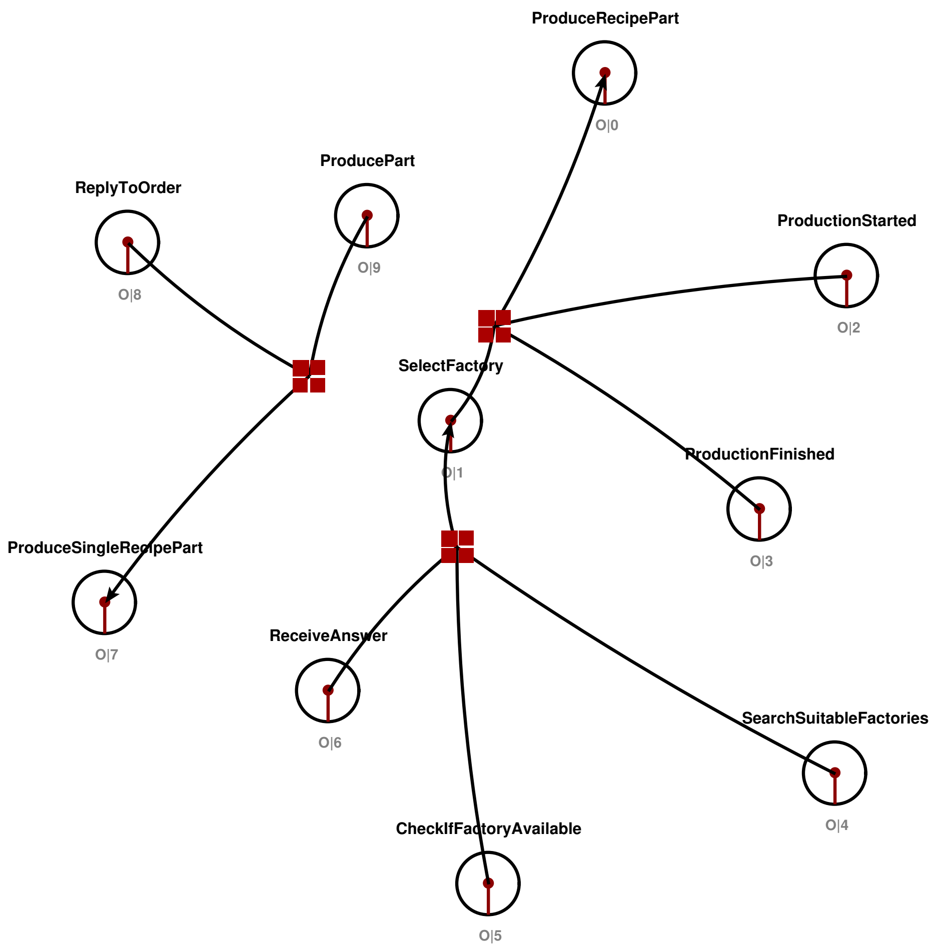


Figure 4.2: The modelled objectives of the *recipeWorld*

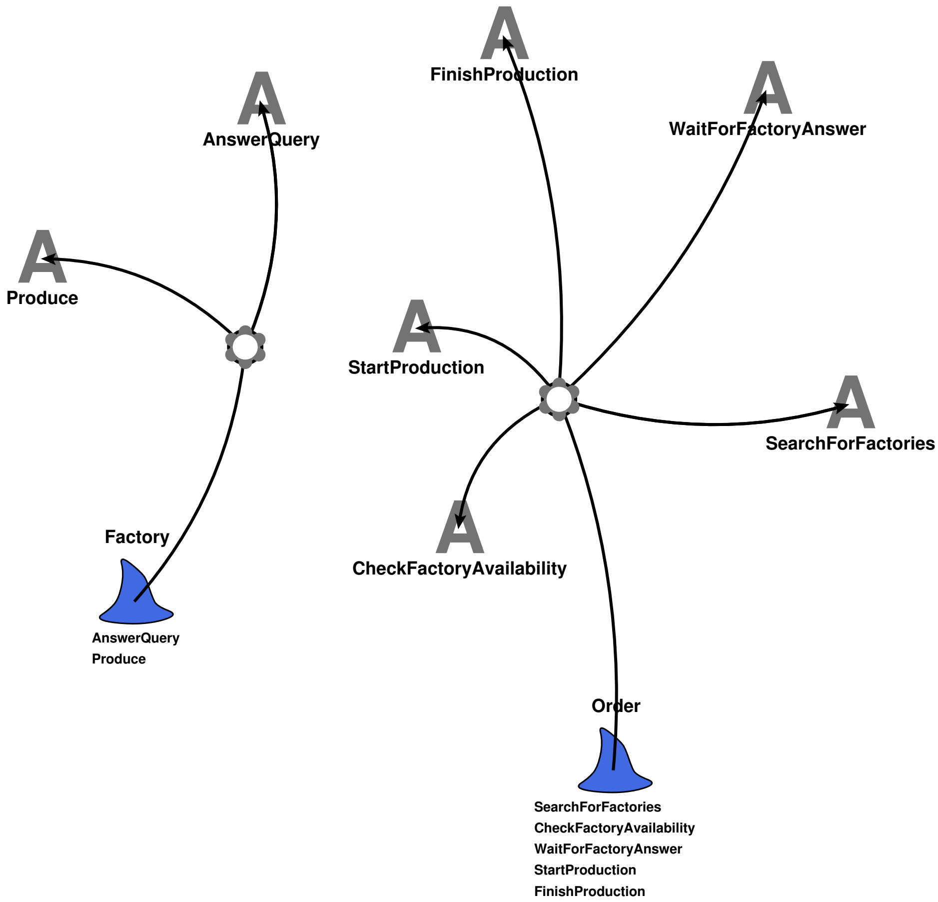


Figure 4.3: The modelled roles, and their actions, of the *recipeWorld*

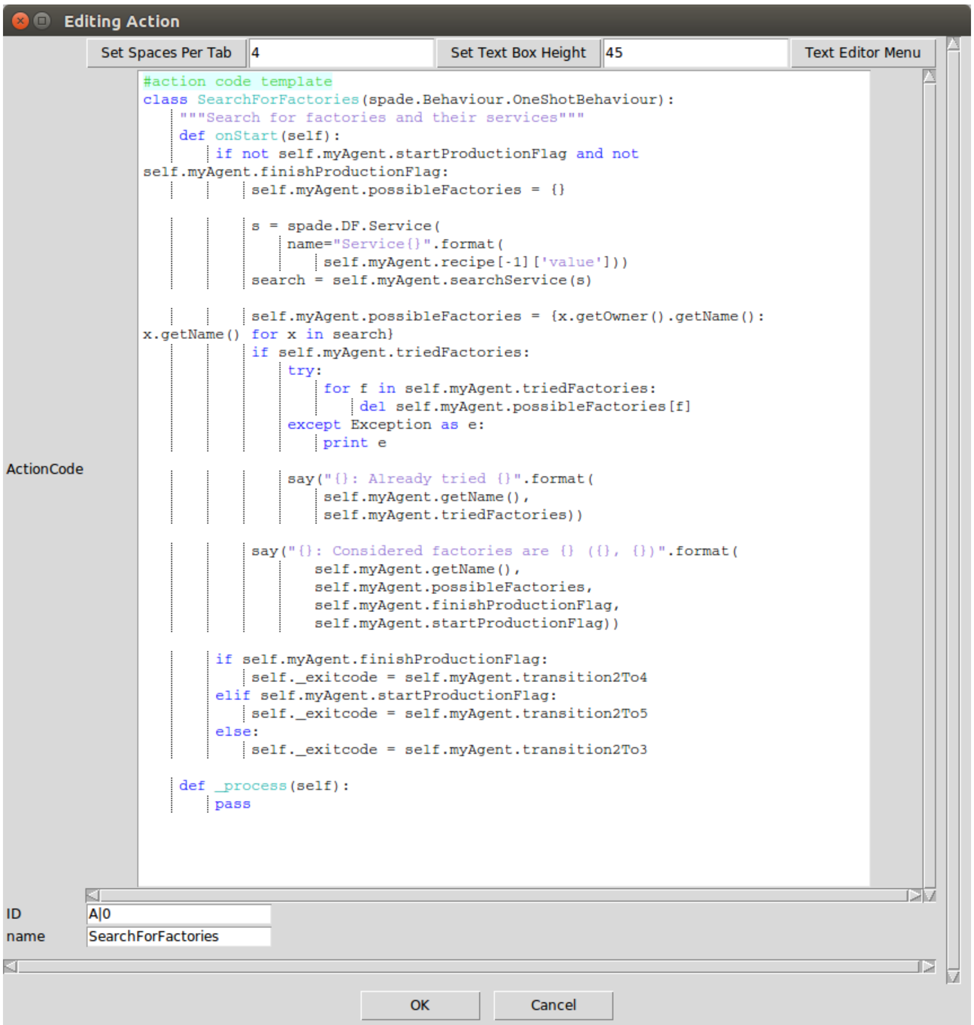


Figure 4.4: Editing attribute values of action *Search-ForFactories*

```
def _setup(self):
    print 'OUOU|0SimpleUnit: running'
    self.addBehaviour(self.ChangeRole(), None)

    self.configureKB('SWI', None, 'swipl')
    self.addBelieve('canHaveRole(OU|0,R|1)')
    self.addBelieve('canHaveRole(OU|0,R|0)')
    self.addBelieve('hasAction(Order,WaitForFactoryAnswer)')
    self.addBelieve('hasAction(Factory,Produce)')
    self.addBelieve('hasAction(Order,FinishProduction)')
    self.addBelieve('hasAction(Order,StartProduction)')
    self.addBelieve('hasAction(Order,CheckFactoryAvailability)')
    self.addBelieve('hasAction(Order,SearchForFactories)')
    self.addBelieve('hasAction(Factory,AnswerQuery)')
    self.addBelieve('canReachGoal(SearchForFactories,SearchSuitableFactories)')
    self.addBelieve('canReachGoal(StartProduction,ProductionStarted)')
    self.addBelieve('canReachGoal(ActionName,ReceiveAnswer)')
    self.addBelieve('canReachGoal(ActionName,ProductionFinished)')
    self.addBelieve('canReachGoal(ActionName,SearchSuitableFactories)')
    self.addBelieve('canReachGoal(WaitForFactoryAnswer,ReceiveAnswer)')
    self.addBelieve('canReachGoal(ActionName,ProductionStarted)')
    self.addBelieve('canReachGoal(FinishProduction,ProductionFinished)')
    self.addBelieve('canReachGoal(Produce,ProducePart)')
    self.addBelieve('canReachGoal(ActionName,ReplyToOrder)')
    self.addBelieve('canReachGoal(CheckFactoryAvailability,CheckIfFactoryAvailable)')
    self.addBelieve('canReachGoal(AnswerQuery,ReplyToOrder)')
    self.addBelieve('canReachGoal(ActionName,ProducePart)')
```

Listing 4.1: Implementation details of *generateNodeCode* function could be named `CommunicateWithFactory`. Such a modelling decision is up to the model developer and the person implementing the system.

The roles defined for this particular example are derived from the initial description of *recipeWorld*. Alternatively, they could be defined using the four-step process presented by Lhaksmana, Murakami and Ishida [73]. First, roles are to be identified based on the available description of the modelled system – it is a system consisting of agents playing as factories and recipes, on their way of creating an interaction network. The defined roles should be then elaborated, and described in detail, including their properties and other identified necessary details. Interaction design step is modified a bit from the description given in [73] – it is less about modelling interaction between agents, agents and roles, or agents and their environment, but is more about modelling behaviours (i.e. actions) that can be used by agents in order to interact with their environment. The final step, assignment, is performed during the modelling process, but can be thought of only as an initial definition of role assignment, since an agent's knowledge base is expected to change while the system is run (a point of view that is described and argued in Section 5.1).

4.2 The Mana World

The Mana World (TMW) is an MMORPG that was used during the Large-Scale Multi-Agent Modelling of Massively On-Line Role-Playing Games (ModelMMORPG) project. A quest was designed specifically for the purposes of the project's research process, which is described in [92, 96].

The quest, named The Quest for the Dragon Egg, demanded players to cooperate, utilise social interaction, and engage in strategic planning. In order to successfully complete the quest, a player has to solve a set of objectives: find the exact location of the Dragon Egg item, retrieve it, transport it to a specific non-player character (NPC), craft a specific item with a rich ingredient list, use it on the Dragon Egg, and visit another specific NPC. The Dragon Egg item can be found in one of the three predefined locations in the in-game world of The Mana World, yet its exact location cannot be known prior to its spawning time (once every 24 hours), and there can never be two usable Dragon Eggs at any given point in time. Each of the specific locations are located in a dragon den, where dragons guard the spawned Dragon Egg item. In order to transport the egg to the designated NPC, three players have to be present at all times, otherwise the egg is dropped, and rendered useless, meaning that the next Dragon Egg spawn must be found.

The described quest is a good example of how MMORPGs emphasise interaction and player cooperation. Further importance of grouping and cooperation is seen in further constraints of the quest, e.g. once a player initiates the mentioned quests, i.e. pickes up the Dragon Egg item, only the members of their group (usually called a party) can complete the quest, and gain the defined rewards (ability to summon a friendly Dragon monster). The key observation in modelling the described quest is the fact that the set of constraints and roles do not change, regardless of the number of individual agents playing the game and solving the quest. The modelled example situation is shown visually in Fig. 4.5.

Concerning the seven organisational perspectives of modelling LSMASs, the built model can be observed as follows.

It is defined by the model that an individual organisational unit (a single player character played by an agent) can be a part of an organisational unit – such a relationship represents party or guild membership. Elements of organisational culture are again portrayed indirectly using the concept of knowledge artefacts – storage of normative elements not included in the definitions of given roles. Speaking of strategies, available actions within the system are defined, and related to specific roles that can be played by individual agents. Furthermore, defined actions have further described affect on the system environment through their connections to the defined objective elements. Organisational dynamics are presumed to be an integral part of the system since a relationship exists between individual organisational unit and a compound organisational unit. More so, a role that can initialise the process of creating compound organisational units is defined, with a part of its role in the organisational dynamics process shown in [125]. Inter-organisational aspects are not present within the example model of this piece of the The Mana World.

TMW example is utilised to exhibit the multimodel modelling. Namely, the main part of the model is modelled as usual, but the `Objective` individuals are loaded from the appropriate database, since several models were saved in advance, one for each quest of interest. Two quest models are shown in Figs. 4.6 and 4.7, while only one of the quests is present in the wholesome model shown in Fig. 4.5.

The noticeable difference in the two modelled quests (sets of objectives) in Figs. 4.6 and 4.7 is their complexity, i.e. the structure of their decomposition. While one is composed of multiple levels of objective grouping, the other is of a simple linear structure. Their main difference, in the context of implementation and application template generation, is that the first quest is generated as a set of shorter plans, while the second one is generated as a plan comprising a longer chain of objectives. This comparison of the two forms of defining objectives represents the level of customised approach provided by Lamrast−+ metamodel and the provided tool.

Furthermore, it is shown here how a general model can be built using several models. Even though the two shown objectives are defined using two separate models that are working using the same database, the application template generating feature is still intact. Additional information for the model was provided using other models, such as the one comprising roles, an organisational unit, and their respective actions. In another model were actions connected to their appropriate objectives. The final result of using

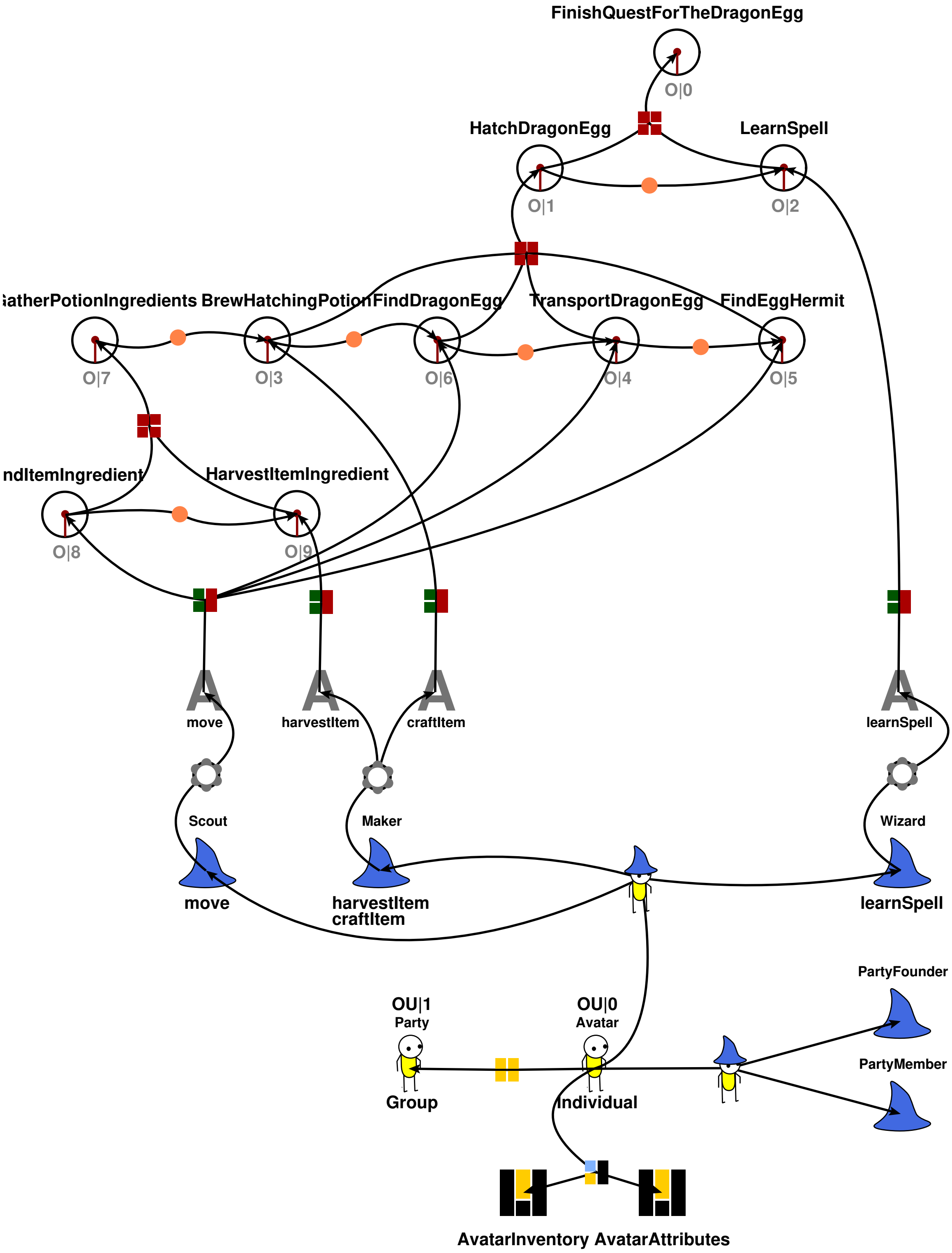


Figure 4.5: The model of the Quest for the Dragon Egg implemented in TMW

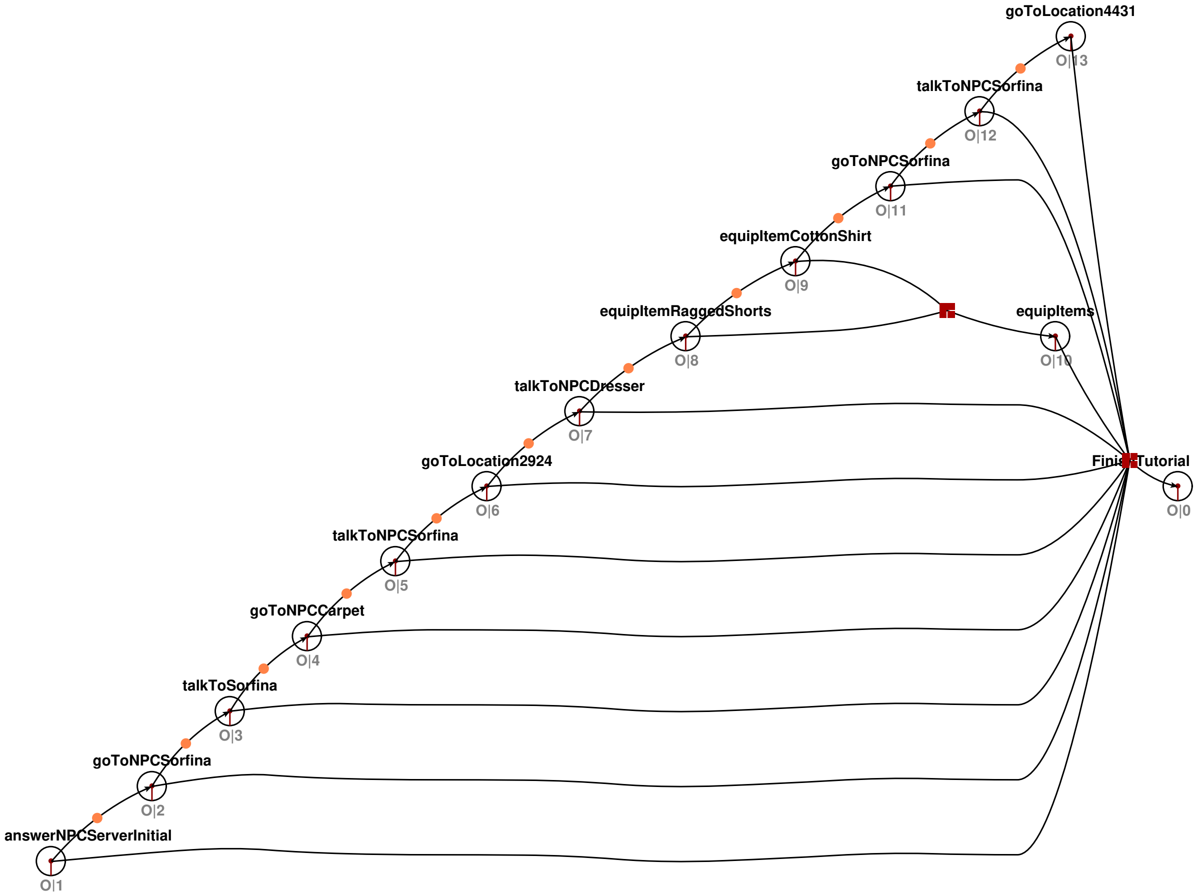


Figure 4.6: Tutorial quest breakdown, from The Mana World

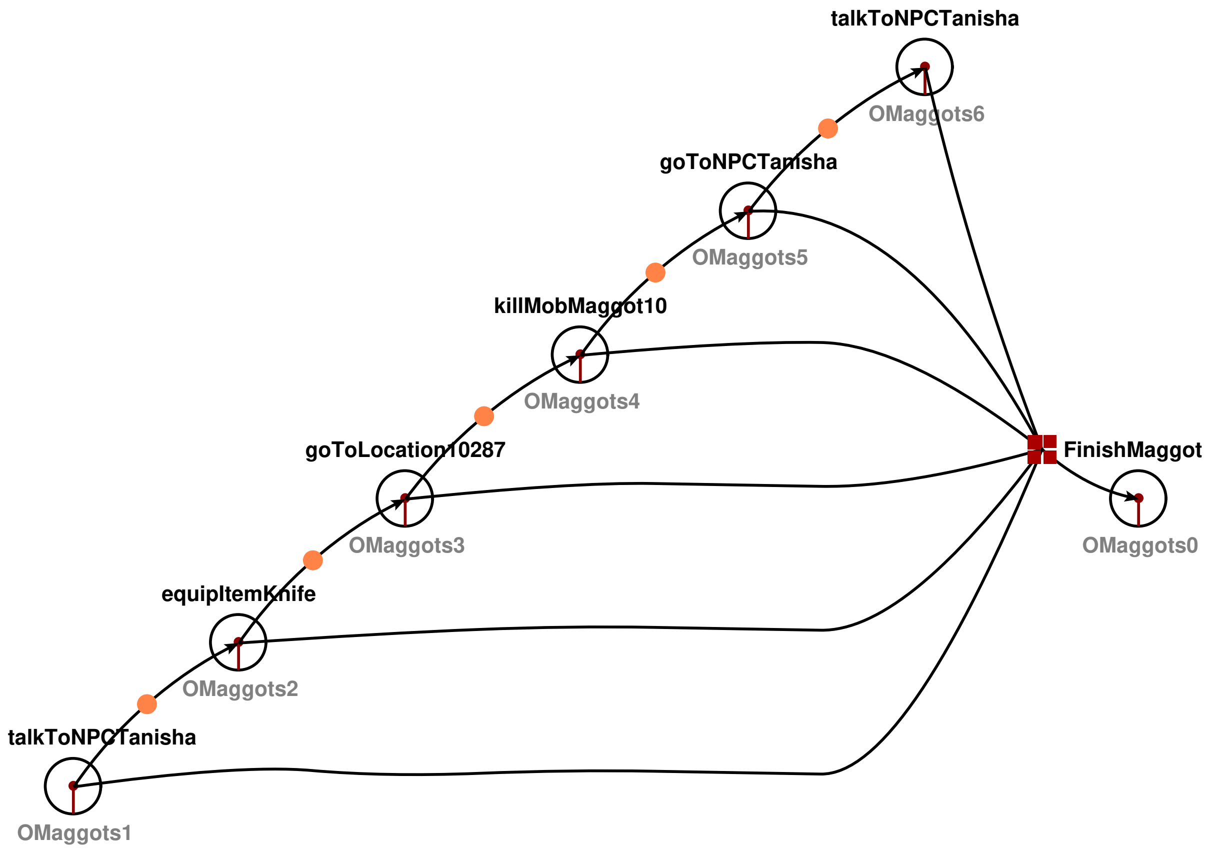


Figure 4.7: A quest breakdown, from The Mana World

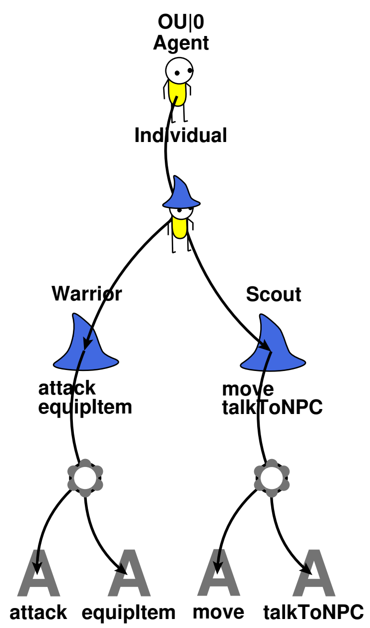


Figure 4.8: Roles and their actions that are used to solve quests from Figs. 4.6 and 4.7, from The Mana World

```
self.configureKB('SWI', None, 'swipl')
self.addBelieve('canHaveRole(OU|0,R|1)')
self.addBelieve('canHaveRole(OU|0,R|0)')
self.addBelieve('hasAction(Scout,talkToNPC)')
self.addBelieve('hasAction(Warrior,attack)')
self.addBelieve('hasAction(Scout,move)')
self.addBelieve('hasAction(Warrior,equipItem)')
self.addBelieve('canReachGoal(equipItem,equipItemRaggedShorts)')
self.addBelieve('canReachGoal(talkToNPC,talkToSorfina)')
self.addBelieve('canReachGoal(equipItem,equipItemKnife)')
self.addBelieve('canReachGoal(move,goToLocation4431)')
self.addBelieve('canReachGoal(equipItem,equipItemCottonShirt)')
self.addBelieve('canReachGoal(talkToNPC,talkToNPCSorfina)')
self.addBelieve('canReachGoal(move,goToNPCCarpet)')
self.addBelieve('canReachGoal(attack,killMobMaggot10)')
self.addBelieve('canReachGoal(talkToNPC,answerNPCServerInitial)')
self.addBelieve('canReachGoal(move,goToNPCSorfina)')
self.addBelieve('canReachGoal(talkToNPC,talkToNPCDresser)')
self.addBelieve('canReachGoal(move,goToNPCTanisha)')
self.addBelieve('canReachGoal(move,goToLocation2924)')
self.addBelieve('canReachGoal(talkToNPC,talkToNPCTanisha)')
self.addBelieve('canReachGoal(move,goToLocation10287)')
```

Listing 4.2: Knowledge base of an organisational unit

the implementation template generating feature is visible the most in an organisational unit's knowledge base, where knowledge of all the modelled roles, actions, and objectives is stored, as shown in Listing 4.2.

The referenced model is available on GitHub repository of the research[2] of this thesis as well. Upon running the application template generator, four files are created or updated if they exist already: two with the initial core code for the modelled organisational units, one with all the available actions and their respective implementation code (where applicable), and one combining both of these files with the basic SPADE system.

4.3 Smart Self-Sustainable Human Settlement with Organisations

The Smart Self-Sustainable Human Settlement (SSSHS) Framework was developed as a part of a PhD research, described in detail by Tomičić [135]. The basic idea of the framework is presented in [136] as follows. A distributed complex self-sustainable system, comprising individual dwelling units, is interconnected in a network that allows those units to exchange both resources and pieces of data. Communication can be initiated when an event leading to one of the two basic scenarios is detected: either resource depletion within a specific subsystem of the observed system, or resource production overflow.

[2]For more information visit `https://github.com/Balannen/LSMASOMM`

Every dwelling unit can be composed of several individual agents, each of them playing one of the specified roles [136]: producer, consumer, or storage. A producer role produces a resource according to the provided input data distribution. The consumer role consumes resources according to the given unit's inner specifics. The storage role is about storing resources, communicating with other units, triggering the predefined self-sustainability mechanisms, etc. Every agent enacting the storage role deals with only one resource type at any given time, and it the main communication point towards other dwelling units, as it communicates with their storage units of the same resource type.

The above described framework does not inherently recognise the concept of an organisation and organisational behaviour, although, it has been studied later, organisational behaviour (forming organisations and coordinated work towards a common goal) may bring benefits to a system comprising smart appliances and similar artificial agents building a smart city.

Using the context of organisation, a dwelling unit can be observed as an organisational unit [136]. Further combined with a defined set of roles, and a specific set of organising criteria (e.g. an objective from the self-sustainability domain, a particular missino, etc.), a higher-level organisational unit is formed. Utilising the recursive definition of an organisatinoal unit featured in Lamrast−+ ontology and metamodel, a dwelling unit can be observed as a higher-level organisational unit when compared with individual appliances (dwelling unit agents that enact either of the three defined roles), or as a lower-level organisational unit, when a group of dwelling units is observed (e.g. flats in a building). Ultimately, each organisational unit, regardless of their observed level, can thus be given a role to play.

The model associated with SSSHS framework shown in Fig. 4.9 is slightly different than that presented in [136], yet the underlying message is the same. It should be noted here that actions modelled in Fig. 4.9 may as well be symbolic, as they can be further developed at the implementation stage, but provide sufficient information at the model level, and serve well the function of visualised system description. The most significant change of the model provided here and the one presented in [136] is that the model in Fig. 4.9 features only two organisational units – one being lower-level and the other being higher-level – as the model can be applied to various levels of grouping – from the individual units, to local grouping, to neighbourhood level, and every organisational unit ultimately plays one of the defined roles, since the framework is defined in such a way.

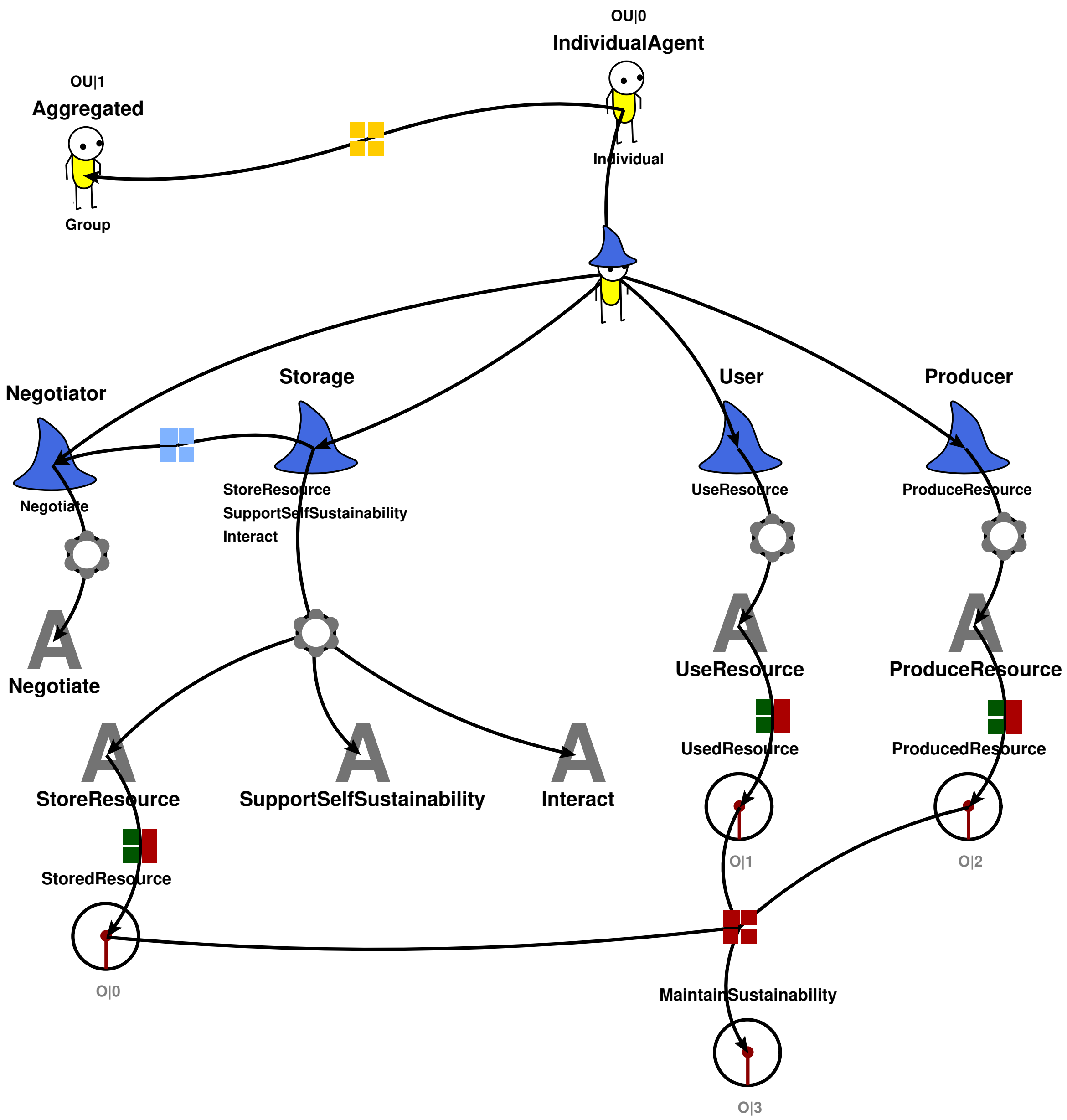


Figure 4.9: SSSHS model

Chapter 5

Conclusion

The following chapter provides a discussion on the developed metamodel and the modelling tool, where both are put into perspective of similar research and discussed in the context of possible improvement. Discussion is followed by the section with concluding remarks which opens some questions and possible further research directions.

5.1 Discussion

Developed on the bases of relevant already published research, this thesis builds its results towards modelling LSMASs and the practical use of the defined Lamrast−+ metamodel. The decision to introduce the practical component to the metamodel definition is based on numerous examples of theoretical developments without the practical element to support their development.

The concepts defined in Lamrast−+ metamodel were chosen from all the elements of the ontology in Section 2.1 for their general application possibilities in the context of MASs and LSMASs, while providing sufficient levels of specificity to the modelled system. Furthermore, since one of the key research objectives is to model organisational concepts applicable to MMORPGs, the selected concepts can be discussed in the context of MMORPGs as well. With organisational units either as single players or groups of players that can join into a higher-level grouping concept that is usually called a *guild*, along with a concept that is quite a standard occurrence in the domain of MMORPGs – a role which is often used to describe a player's avatar's position in the social and power structure of the in-game world, or their sets of abilities, skills, and character traits, coupled with role-dependent actions, and quests in terms of structured sets of objectives – applicability of the model in the domain of MMORPGs is obvious. However, the constrained expressiveness can be perceived as a weakness, since further domain elements should be introduced for an even clearer domain description that would provide further details about an observed system. Concepts that would directly allow for such more detailed description of the given application domain were not considered here for their inclusion would not benefit the system immensely, yet the sense of metamodel's applicability to application domains other than MMORPGs would certainly suffer. Ultimately, the goal of this research was not to create an extremely domain-specific metamodel, but to make it applicable to various other domains, such as the Internet of Things and similar, as well.

One of the challenges of the research was therefore whether the defined model is in fact a model or a metamodel, since it has many possible forms or domains of application. The arguments in favour of the metamodel concept are laid out throughout this thesis, yet if Lamrast−+ metamodel is used to model the system directly, not taking care of the implementation, and serving as a tool of describing a given system, the metamodel may as well be named a model, since its use represents a specific system directly. In other words, should the example described in Section 2.2.2, which features two specific players (actually, their avatars), and specific quests, and roles, be described directly and in its entire specifics, using the concepts defined in Lamrast−+ metamodel, the use resembles more that of a model. On the other hand, when the metamodel is used to describe a system, but remains on a certain level of abstraction, e.g. defining a large number of individual agents as simply and organisational unit, the resulting model (where further instantiating is necessary before agents themselves are reached), is more similar to the metamodel-model relationship, as opposed to being a mere model. A further argument in favour of the model concept lies in the fact that the model level (which describes an observed system) provides the model developer with the opportunity to include specific programming code in some of the elements' attributes – thus, the element is set as a direct representation of a specific real object (e.g. an action performable by an agent that can be implemented using the programming code provided as its attribute value). Still, if the model is provided with features as described in this here paragraph, than the model defining the used elements and their features and rules of their connections, is a metamodel, and Lamrast−+ metamodel is exactly that – it provides the definitions and rules of use for the concepts that are used to describe a specific system comprising agents.

Since complex systems are prone to having complex representations, although not necessarily actually having them, the multimodel modelling is a welcome addition to the practical application possibilities of Lamrast−+ metamodel. The differences and similarities of such an approach and the multi-perspective approach were discussed in Section 3.2. Saving and restoring model elements independently from the saved model itself is beneficial for the purposes of recreating a view of the given model without having to open the model itself. Furthermore, since elements of a model can be restored independently of their immediate neighbours, it is possible to create a big model containing most of the necessary elements, and construct the wanted views afterwards. Furthermore, if an element is used often, it may be saved (or a set of elements) in a specific template-based model database, and restored later when needed. Certainly, this approach has its weaknesses, such as a great possibility for developing irregularities or invalid situations when elements are saved or loaded. Furthermore, saving and loading is always performed within a specific context, and the saving and the loading context can be significantly different. The modelling tool takes care of some of the aspects of context differences at the moment, yet further work should be done to further smooth out user experience. The approach as described in this paragraph is especially important in the context of recursive definition of an organisational unit, like the one used in this thesis, described by Schatten [118].

The application template generating feature is a welcome addition to the modelling tool and provides the model developer with an appreciated feature. The generated programming code greatly depends on how studious was the model developer, and with how much information they provided the model. Furthermore, how the model developer perceives the generated application template depends on their initial expectations when programming code generation is considered. Although the application template provided is almost enough for a system to be run, it is still only a template, a skeleton of sort, and requires substantial further development, if the system is to be implemented according to the model developer's expectations. The current version of the modelling tool provides only simple, proof-of-concept features of application template generation, yet it clearly shows the potential of its development. The current constraint of implementing only a system using the specific MASs development platform can be mitigated by including further implementation options.

Organisational unit individual's knowledge about the modelled system, i.e. its knowledge base, as described in Section 3.3.1, is embedded into the definition of an agent, thus being defined as its default knowledge. Yet, knowledge of a SPADE agent is not static, and can be modified through time. Since each individual agent's knowledge base can be modified individually, the modelled system can be defined with only a starting set of organisational constraints, some of which, that are stored in individual agent's knowledge bases, can be modified depending on the activity within the system and the behaviour of individual agents. Therefore, using this feature of agent's knowledge being modified at runtime, it can be said that Lamrast−+ metamodel, coupled with SPADE implementation platform for MASs , can be used for modelling and running complex self-organising systems. Customisable knowledge base implies dynamic role enactment in this context.

Roles modelled using Lamrast−+ metamodel are defined as a kind of normative constraint groups that enable agents to play specific actions towards achieving specific atomic goals. Roles are not defined on the organisational unit's basis, nor are they strictly coupled with specific organisational units. Rather, roles are defined as existent in the modelled system, and are implemented as behaviours disposable to organisational units. The modelled association of `canHaveRole` defines roles that an organisational unit can play at the implemented system's initiation, as a stored piece of knowledge in organisational units. When the system is run, this set of roles that can be played by an organisational unit can be changed if an organisational unit learns about an action of a role the knowledge of which it did not have before. Such behaviours (i.e. actions) and roles have to be defined beforehand, while the system is being implemented. Such an approach is in accordance with the lack of features in existing methodologies for development of MASs described by Lhaksmana, Murakami and Ishida [73]:

> "To model self-organizing MAS with such capability, MAS designers should be able to design how the agent will adapt itself instead of defining a set of fixed functionalities at design time. Another required feature for designing self-organizing MAS is the separation between designing agent behaviors and agent behavior adaptation. The former means designing the actions that can be performed by the agents, whereas the latter means defining which actions that can (or cannot) be performed in which situations." — Lhaksmana, Murakami and Ishida [73]

When Lamrast−+ metamodel is compared to the modern example model bent on modelling self-organising MASs, described by Lhaksmana, Murakami and Ishida [73], it is easy to observe that both models have the `Role` concept at their core, probably since *a role usually represents a set of functionalities, a position of duty or an aggregation of behaviors to be played by agents* [73]. Lamrast−+ metamodel implements the `Role` concept using the third offered definition of a role in MASs, possibly coupled with the first one. Furthermore, four activities towards modelling roles: 1) identification, 2) elaboration, 3) interaction design, 4) assignment, can be followed when roles are modelled using Lamrast−+ metamodel, as shown in Chapter 4. As opposed to the role modelling metamodel proposed by Lhaksmana, Murakami and Ishida [73], Lamrast−+ metamodel is not as complex and detailed when roles are considered, since Lamrast−+ metamodel aims at modelling a wider set of concepts, and in less detail, for the necessary details are expected to be implemented alongside the detailed system implementation process. Further metamodels for modelling LSMASs do exist, as presented in Section 1.4.3.

Since Lamrast−+ metamodel is defined as a rather general one, it is possible to develop extensions to its concepts, thus making it more specific for a given domain, or more customised for a specific purpose.

Chapter 4 describes three examples that are modelled using the concepts of Lamrast−+ metamodel, and how the features of multimodel modelling and application template generator work. The examples are chosen from multiple application domains, as opposed to only a single one (e.g. MMORPGs), in order to show the diversity of application domains modelling whereof the metamodel can be used. The metamodel is therefore showcased on a broader spectrum of application domains than initially intended, as defined by one of the key research questions. Although the benefit of having a generally applicable metamodel is a benefit in itself, it certainly is a disadvantage in the context of reduced expressiveness of the model modelled using Lamrast−+ metamodel. The fine line between the two can be bridged during the implementation phase.

In accordance with what was mentioned earlier in this thesis (namely in Section 2.2), Lamrast−+ metamodel does not stand alone in the set of available models for modelling LSMASs, nor indeed in the context of organisational modelling of LSMASs. The main improvement upon those other available models is the level at which Lamrast−+ metamodel conforms to the seven perspectives of organisational modelling of LSMASs laid out by Schatten [118], its efficient combination of organisational concepts with concepts applicable to LSMASs and intelligent virtual environments (IVEs), and the available modelling tool where the metamodel can be used.

5.2 Future Research

The purpose of research is not only to provide answers to existing questions, but to uncover some new challenges that can be engaged in and dealt with.

Aside from regular improvements in the terms of programming code optimisation or visual formatting of the modelling tool, some further groundwork can be performed for an even better metamodel, and the accompanying modelling tool.

It was mentioned in Section 2.2.1.4 that the `Objective` concept of the metamodel can be defined using a number of attributes, two of which are not included in the application template generator – `Reward` and `Measurement`. These attributes are interesting concepts for future research, since they would provide model developers with even greater modelling possibilities and automatising of the modelled system's development process. Such a development would demand further improvements in knowledge bases pertaining the modelled system – those of individual agents, as well as those available in the system that are not initially accessible to the system's agents, but have to be discovered. Such an idea is in complete accordance with the context of LSMASs.

Development and improvement of the knowledge management process supported by the metamodel, and by succession its modelling tool, would prove useful as well. A part of the system knowledge that should be modelled is organisational culture – a mixture of all kinds of knowledge from various domains and of various importance – which is, at the moment, modelled using non-detailed concepts of knowledge artefacts, which offer a myriad of opportunities for further research.

One possibility for tackling the knowledge management perspective is knowledge storage in an ontology accessible to organisational units. Such an approach might foster the process of reasoning to individual organisational units, although selective knowledge access may prove challenging. Nonetheless, since one of the key aspects of ontologies is knowledge sharing, this development direction may prove beneficial.

The developed metamodel, and the accompanying ontology, can always be improved, especially when the metamodel is applied to further application domains of LSMASs. Enhancement of the metamodel and the ontology is foreseen in the context of specialisation as well, as opposed to keeping them on the current level of abstraction only. In the context of computer games, a more specific ontology that could be used for a more expressive description of a given domain (i.e. a computer game), is deemed as beneficial as it may provide a new approach to modelling MASs applicable to that particular application domain.

Paired with an implemented application programming interface (API) for a specific computer game, the metamodel, and especially the modelling tool, may be modified insomuch as to provide assistance in development of MASs that contain all the actions necessary for agents to start playing a given game. Such a combination would provide game developers with the ability to test their games logic- and story-wise, apart from the currently available load-based testing only. API for TMW is one of the future steps planned as a research extending that of ModelMMORPG project.

After all, a doctoral thesis and the accompanying research are only an introduction.

Bibliography

[1] H. A. Abbas. 'Exploiting the Overlapping of Higher Order: Entities Within Multi-Agent Systems'. In: *International Journal of Agent Technologies and Systems* 6.3 (July 2014), pp. 32–57. ISSN: 1943-0744. DOI: `10.4018/ijats.2014070102`.

[2] H. A. Abbas. 'Realizing the NOSHAPE MAS Organizational Model:' in: *International Journal of Agent Technologies and Systems* 7.2 (Apr. 2015), pp. 75–104. ISSN: 1943-0744. DOI: `10.4018/IJATS.2015040103`.

[3] H. A. Abbas, S. I. Shaheen and M. H. Amin. 'Organization of Multi-Agent Systems: An Overview'. In: *International Journal of Intelligent Information Systems* 4.3 (2015), p. 46. ISSN: 2328-7675. DOI: `10.11648/j.ijiis.20150403.11`.

[4] E. Argente et al. 'Supporting Agent Organizations'. In: *Multi-Agent Systems and Applications V*. Ed. by H.-D. Burkhard et al. Lecture Notes in Computer Science 4696. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Chap. 24, pp. 236–245. ISBN: 978-3-540-75254-7. DOI: `10.1007/978-3-540-75254-7_24`.

[5] S. Assar. 'Meta-Modeling: Concepts, Tools and Applications'. In: *IEEE RCIS '15 : 9th International Conference on Research Challenges in Information Science*. 9th International Conference on Research Challenges in Information Science. Athens, Greece: IEEE, 2015.

[6] C. Atkinson and T. Kühne. 'Model-Driven Development: A Metamodeling Foundation'. In: (2003), pp. 1–7.

[7] L. Atzori, A. Iera and G. Morabito. 'The Internet of Things: A Survey'. In: *Computer Networks* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 13891286. DOI: `10.1016/j.comnet.2010.05.010`.

[8] A. Barella et al. 'MAM5: Multi-Agent Model for Intelligent Virtual Environments'. In: *10th European Workshop on Multi-Agent Systems (EUMAS 2012)*. 2012, pp. 16–30.

[9] F. Béhé et al. 'An Ontology-Based Metamodel for Multiagent-Based Simulations'. In: *Simulation Modelling Practice and Theory* 40 (2014), pp. 64–85. ISSN: 1569190X. DOI: `10.1016/j.simpat.2013.09.002`.

[10] C. Bernava et al. ‘RDF Annotation of Second Life Objects: Knowledge Representation Meets Social Virtual Reality’. In: *Computational and Mathematical Organization Theory* 20.1 (Mar. 2014), pp. 20–35. ISSN: 1381-298X, 1572-9346. DOI: 10.1007/s10588-012-9148-4. arXiv: 1504.02358.

[11] J. Bezivin and O. Gerbe. ‘Towards a Precise Definition of the OMG/MDA Framework’. In: *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*. San Diego, CA, USA: IEEE Computer Society, 2001, pp. 273–280. ISBN: 0-7695-1426-X. DOI: 10.1109/ASE.2001.989813.

[12] L. Birdsey, C. Szabo and K. Falkner. ‘Identifying Self-Organization and Adaptability in Complex Adaptive Systems’. In: *2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, 2017, pp. 131–140. ISBN: 978-1-5090-6555-4. DOI: 10.1109/SASO.2017.22.

[13] L. Birdsey, C. Szabo and K. Falkner. ‘Large-Scale Complex Adaptive Systems Using Multi-Agent Modeling and Simulation’. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. São Paulo, Brazil: International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 1478–1480.

[14] G. Boella, L. van der Torre and H. Verhagen. ‘Introduction to Normative Multiagent Systems’. In: *Computational & Mathematical Organization Theory* 12.2-3 (2006), pp. 71–79. ISSN: 1862-4405. DOI: 10.1007/s10588-006-9537-7.

[15] R. Boero et al. *Agent-Based Models of the Economy: From Theories to Applications*. Palgrave Macmillan, 2015. 232 pp. ISBN: 978-1-137-33980-5.

[16] J. Boes and F. Migeon. ‘Self-Organizing Multi-Agent Systems for the Control of Complex Systems’. In: *Journal of Systems and Software* 134 (Dec. 2017), pp. 12–28. ISSN: 01641212. DOI: 10.1016/j.jss.2017.08.038.

[17] D. Bork et al. ‘Conceptual Modelling for Smart Cities : A Teaching Case’. In: *Smart City Learning: Opportunities and Challenges* 27.1 (2015), pp. 10–28. ISSN: 22832998 18269745.

[18] D. Bork et al. ‘Using Conceptual Modeling to Support Innovation Challenges in Smart Cities’. In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, Dec. 2016, pp. 1317–1324. ISBN: 978-1-5090-4297-5. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0187.

[19] J. C. Burguillo. ‘Self-Organization’. In: *Self-Organizing Coalitions for Managing Complexity*. Emergence, Complexity and Computation 29. Cham: Springer International Publishing, 2018. Chap. 6, pp. 89–100. ISBN: 978-3-319-69898-4. DOI: 10.1007/978-3-319-69898-4_6.

[20] J. C. Burguillo. *Self-Organizing Coalitions for Managing Complexity*. Vol. 29. Emergence, Complexity and Computation. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-69896-0. DOI: 10.1007/978-3-319-69898-4.

[21] C. Cameron et al. 'Using Self-Organizing Architectures to Mitigate the Impacts of Denial-of-Service Attacks on Voltage Control Schemes'. In: *IEEE Transactions on Smart Grid* (2018), pp. 1–1. ISSN: 1949-3053. DOI: 10.1109/TSG.2018.2817046.

[22] K. M. Carley and L. Gasser. 'Computational Organization Theory'. In: *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Ed. by G. Weiss. Cambridge, MA, USA: MIT Press, 1999. Chap. Computatio, pp. 299–330. ISBN: 0-262-23203-0.

[23] S. Čaušević, M. Warnier and F. M. Brazier. 'Dynamic, Self-Organized Clusters as a Means to Supply and Demand Matching in Large-Scale Energy Systems'. In: *Proceedings of the 2017 IEEE 14th International Conference on Networking, Sensing and Control, ICNSC 2017* (2017), pp. 568–573. DOI: 10.1109/ICNSC.2017.8000154.

[24] M.-H. Chang and J. E. Harrington Jr. 'Agent-Based Models of Organizations'. In: *Handbook of Computational Economics*. Ed. by L. Tesfatsion and K. L. Judd. 1st ed. Vol. 2. Amsterdam, NL: Elsevier, 2006, pp. 1273–1337. ISBN: 978-0-444-51253-6. DOI: 10.1016/S1574-0021(05)02026-5.

[25] P. P. Chen et al. *Conceptual Modeling: Current Issues and Future Directions*. Ed. by P. P. Chen et al. Vol. 1565. Lecture Notes in Computer Science 1565. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. 316 pp. ISBN: 978-3-540-65926-6. DOI: 10.1007/3-540-48854-5.

[26] J. S. Coleman. *Foundations of Social Theory*. Harvard University Press, 1998. 993 pp. ISBN: 978-0-674-31226-5.

[27] D. D. Corkill and S. E. Lander. 'Diversity in Agent Organizations'. In: *Object Magazine* 8.4 (1998), pp. 41–47.

[28] A. Corradini et al. 'Algebraic Approaches to Graph Transformation - Part 1: Basic Concepts and Double Pushout Approach'. In: *Handbook of Graph Grammars and Computing by Graph Transformation*. Ed. by G. Rozenberg. Singapore: World Scientific, 1997. Chap. 3, pp. 163–245. DOI: 10.1142/9789812384720_0003.

[29] L. R. Coutinho, J. S. Sichman and O. Boissier. 'Modelling Dimensions for Agent Organizations'. In: *Handbook of Research on Multi-Agent Systems*. Ed. by V. Dignum. IGI Global, 2009, pp. 18–50. ISBN: 978-1-60566-256-5. DOI: 10.4018/978-1-60566-256-5.ch002.

[30] R. L. Daft. *Organization Theory and Design*. 10th ed. Cengage Learning, 2010. ISBN: 978-1-133-46394-8.

[31] T. De Wolf and T. Holvoet. 'Emergence Versus Self-Organisation: Different Concepts but Promising When Combined'. In: *Engineering Self-Organising Systems*. International Workshop on Engineering Self-Organising Applications. Ed. by S. A. Brueckner et al. Red. by D. Hutchison et al. Lecture Notes in Computer Science 3464. Berlin, Heidelberg: Springer, 2005, pp. 1–15. ISBN: 978-3-540-31901-6. DOI: `10.1007/11494676_1`.

[32] J. de Lara and H. Vangheluwe. 'Using AToM as a Meta-CASE Tool'. In: *ICEIS*. Vol. 2. 2002, pp. 642–649.

[33] J. de Lara and H. Vangheluwe. 'Using Meta-Modelling and Graph Grammars to Process GPSS Models'. In: *The European Simulation Multi-Conference*. 2002, pp. 100–107.

[34] K. S. Decker. 'TÆMS : A Framework for Environment Centered Analysis & Design of Coordination Mechanisms'. In: *Foundations of Distributed Artificial Intelligence*. Ed. by G. M. P. O'Hare and N. R. Jennings. New York, NY, USA: John Wiley & Sons, Inc, 1996. Chap. 16, pp. 429–448. ISBN: 0-471-00675-0. DOI: `10.1.1.45.8925`.

[35] V. Dignum. 'A Model for Organizational Interaction: Based on Agents, Founded in Logic'. Doctoral thesis. Utrecht University, 2004.

[36] V. Dignum. 'The Role of Organization in Agent Systems'. In: *Handbook of Research on Multi-Agent Systems*. Ed. by V. Dignum. IGI Global, 2009, pp. 1–16. ISBN: 978-1-60566-256-5. DOI: `10.4018/978-1-60566-256-5.ch001`.

[37] H. Ehrig. 'Introduction to the Algebraic Theory of Graph Grammars (a Survey)'. In: *Graph-Grammars and Their Application to Computer Science and Biology*. Ed. by V. Claus, H. Ehrig and G. Rozenberg. Lecture Notes in Computer Science 73. Berlin, Heidelberg: Springer, 1979. Chap. 1, pp. 1–69. ISBN: 978-3-540-09525-5. DOI: `10.1007/BFb0025714`.

[38] D. W. Embley and B. Thalheim, eds. *Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges*. Berlin, Heidelberg: Springer, 2011. ISBN: 978-3-642-15864-3. DOI: `10.1007/978-3-642-15865-0`.

[39] J. Engelfriet and G. Rozenberg. 'Node Replacement Graph Grammars'. In: *Handbook of Graph Grammars and Computing by Graph Transformation*. Ed. by G. Rozenberg. River Edge, NJ, USA: World Scientific Publishing, 1997. Chap. 1, pp. 1–94. ISBN: 981-02-2884-8.

[40] M. Esteva, J. Padget and C. Sierra. 'Formalizing a Language for Institutions and Norms'. In: *Intelligent Agents VIII*. Ed. by J.-J. C. Meyer and M. Tambe. Lecture Notes in Computer Science 2333 2333. Berlin, Heidelberg: Springer, 2002, pp. 348–366. ISBN: 978-3-540-45448-9. DOI: `10.1007/3-540-45448-9_26`.

[41] J. Ferber, O. Gutknecht and F. Michel. 'From Agents to Organizations: An Organizational View of Multi-Agent Systems'. In: *Agent-Oriented Software Engineering (AOSE) IV*. Ed. by P. Giorgini, J. P. Müller and J. Odell. Red. by G. Goos, J. Hartmanis and J. van Leeuwen. Vol. 2935. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 214–230. ISBN: 978-3-540-24620-6. DOI: `10.1007/978-3-540-24620-6_15`.

[42] M. Fernández-López, A. Gómez-Pérez and N. Juristo. 'METHONTOLOGY: From Ontological Art Towards Ontological Engineering'. In: *AAAI-97 Spring Symposium Series* SS-97-06 (1997), pp. 33–40. DOI: `10.1109/AXMEDIS.2007.19`.

[43] M. Fontana and P. Terna. *From Agent-Based Models to Network Analysis (and Return): The Policy-Making Perspective*. 201507. Torino, IT: Department of Economics and Statistics "Cognetti de Martiis", University of Turin, Jan. 2015, pp. 1–19.

[44] A. Freitas et al. 'Semantic Representations of Agent Plans and Planning Problem Domains'. In: *Engineering Multi-Agent Systems*. Ed. by F. Dalpiaz, J. Dix and M. B. van Riemsdijk. Vol. 8758. Cham: Springer International Publishing, 2014, pp. 351–366. ISBN: 978-3-319-14483-2. DOI: `10.1007/978-3-319-14484-9_18`.

[45] L. W. Friedman. *The Simulation Metamodel*. 1st ed. Boston, MA, USA: Springer, 1996. ISBN: 978-1-4612-8556-4. DOI: `10.1007/978-1-4613-1299-4`.

[46] L. Gasser. 'Perspectives on Organizations in Multi-Agent Systems'. In: *Multi-Agent Systems and Applications*. Ed. by M. Luck et al. Red. by G. Goos, J. Hartmanis and J. van Leeuwen. Vol. 2086. Lecture Notes in Computer Science, Vol 2086. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. Chap. 1, pp. 1–16. ISBN: 978-3-540-42312-6. DOI: `10.1007/3-540-47745-4_1`.

[47] A. S. Gazafroudi et al. 'Organization-Based Multi-Agent System of Local Electricity Market: Bottom-Up Approach'. In: *Trends in Cyber-Physical Multi-Agent Systems. The PAAMS Collection - 15th International Conference, PAAMS 2017*. Ed. by F. D. la Prieta et al. 1st ed. Advances in Intelligent Systems and Computing 619. Cham: Springer International Publishing, 2018. Chap. 38, pp. 281–283. DOI: `10.1007/978-3-319-61578-3_38`.

[48] B. Goertzel et al. *Real-World Reasoning: Toward Scalable, Uncertain Spatiotemporal, Contextual and Causal Inference*. 1st ed. Atlantis Thinking Machines 2 2. Atlantis Press, 2011. 269 pp. ISBN: 978-94-91216-10-7. DOI: `10.2991/978-94-91216-11-4`.

[49] A. Gómez-Pérez, M. Fernández and A. J. de Vicente. 'Towards a Method to Conceptualize Domain Ontologies'. In: *Proceedings Workshop: Ontological Engineering*. European Conference on Artificial Intelligence. Budapest, HU: Facultad de Informática (UPM), 1996, pp. 41–51. DOI: 10.1.1.24.167.

[50] A. Gómez-Pérez, N. Juristo and J. Pazos. 'Evaluation and Assessment of Knowledge Sharing Technology'. In: *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*. Ed. by N. J. I. Mars. IOS Press, 1995. Chap. 29, pp. 289–296.

[51] V. Grimm and S. F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, 2005. 448 pp. ISBN: 978-0-691-09666-7.

[52] T. R. Gruber. 'A Translation Approach to Portable Ontology Specifications'. In: *Knowledge Acquisition* 5.2 (1993), pp. 199–220. ISSN: 1042-8143. DOI: 10.1006/knac.1993.1008.

[53] G. Guizzardi. 'On Ontology, Ontologies, Conceptualizations, Modeling Languages, and (Meta)Models'. In: *Proceedings of the 2007 Conference on Databases and Information Systems IV: Selected Papers from the Seventh International Baltic Conference DB&IS'2006*. International Baltic Conference. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2007, pp. 18–39. ISBN: 978-1-58603-715-4.

[54] G. Guizzardi. 'Ontological Foundations for Conceptual Modeling with Applications'. Doctoral thesis. Enschede, NL: University of Twente, 2005. 416 pp.

[55] A. Habel, J. Müller and D. Plump. 'Double-Pushout Graph Transformation Revisited'. In: *Mathematical Structures in Computer Science* 11.5 (2001), pp. 637–688. ISSN: 0960-1295. DOI: 10.1017/S0960129501003425.

[56] R. Hadfi and T. Ito. 'Holonic Multiagent Simulation of Complex Adaptive Systems'. In: *Highlights of Practical Applications of Scalable Multi-Agent Systems. The PAAMS Collection*. International Conference on Practical Applications of Agents and Multi-Agent Systems. Ed. by J. Bajo et al. Communications in Computer and Information Science 616. Cham, CH: Springer, 2016, pp. 137–147. ISBN: 978-3-319-39387-2. DOI: 10.1007/978-3-319-39387-2_12.

[57] M. Hadzic et al. *Ontology-Based Multi-Agent Systems*. Studies in Computational Intelligence 219 219. Berlin, Heidelberg: Springer, 2009. ISBN: 978-3-642-01903-6. DOI: 10.1007/978-3-642-01904-3.

[58] F. Harary. *Graph Theory*. Advanced Book Program 2787 2787. Addison-Wesley, 1969. 274 pp. ISBN: 978-0-201-41033-4. arXiv: 1102.1087.

[59] B. Henderson-Sellers. *On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages*. SpringerBriefs in Computer Science. Berlin, Heidelberg: Springer, 2012. 106 pp. ISBN: 978-3-642-29824-0. DOI: 10.1007/978-3-642-29825-7.

[60] B. Horling and V. Lesser. 'Quantitative Organizational Models for Large-Scale Agent Systems'. In: *Massively Multi-Agent Systems I*. First International Workshop on Massively Multi-Agent Systems. Ed. by T. Ishida, L. Gasser and H. Nakashima. Lecture Notes in Computer Science 3446. Berlin, Heidelberg: Springer, 2005. Chap. Quantitati, pp. 121–135. ISBN: 978-3-540-31889-7. DOI: 10.1007/11512073_9.

[61] J. F. Hübner, J. S. Sichman and O. Boissier. 'A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems'. In: *Advances in Artificial Intelligence*. Brazilian Symposium on Artificial Intelligence. Ed. by G. Bittencourt and G. L. Ramalho. Lecture Notes in Computer Science 2507. Berlin, Heidelberg: Springer, 2002, pp. 118–128. ISBN: 978-3-540-36127-5. DOI: 10.1007/3-540-36127-8_12.

[62] J. F. Hübner, L. Vercouter and O. Boissier. 'Instrumenting Multi-Agent Organisations with Artifacts to Support Reputation Processes'. In: *Coordination, Organizations, Institutions and Norms in Agent Systems IV*. International Workshop on Coordination, Organization, Institutions and Norms in Agent Systems. Ed. by J. F. Hübner et al. Lecture Notes in Computer Science 5428. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 96–110. ISBN: 978-3-642-00443-8. DOI: 10.1007/978-3-642-00443-8_7.

[63] R. Iqbal et al. 'An Analysis of Ontology Engineering Methodologies: A Literature Review'. In: *Research Journal of Applied Sciences, Engineering and Technology* 6.16 (2013), pp. 2993–3000. ISSN: 20407459.

[64] D. Karagiannis. 'Agile Modeling Method Engineering'. In: *Proceedings of the 19th Panhellenic Conference on Informatics*. Panhellenic Conference on Informatics. New York, NY, USA: ACM Press, 2015, pp. 5–10. ISBN: 978-1-4503-3551-5. DOI: 10.1145/2801948.2802040.

[65] D. Karagiannis, H. C. Mayr and J. Mylopoulos, eds. *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*. Cham, CH: Springer, 2016. ISBN: 978-3-319-39416-9. DOI: 10.1007/978-3-319-39417-6.

[66] D. Karagiannis et al. 'Fundamental Conceptual Modeling Languages in OMiLAB'. In: *Domain-Specific Conceptual Modeling*. Ed. by D. Karagiannis, H. C. Mayr and J. Mylopoulos. 1st ed. Cham, CH: Springer, 2016, pp. 3–30. DOI: 10.1007/978-3-319-39417-6_1.

[67] S. A. Kidanu, R. Chbeir and Y. Cardinale. 'MAS2DES-Onto: Ontology for MAS-Based Digital Ecosystems'. In: *Simposio Latinoamericano de Manejo de Datos e Información (SLMDI) - JAIIO 46*. Simposio Latinoamericano de Manejo de Datos e Información. Córdoba, AR: Sociedad Argentina de Informática e Investigación Operativa (SADIO), 2017.

[68] C. Kiourt and D. Kalles. 'A Platform for Large-Scale Game-Playing Multi-Agent Systems on a High Performance Computing Infrastructure'. In: *Multiagent and Grid Systems* 12.1 (2016), pp. 35–54. ISSN: 15741702. DOI: 10.3233/MGS-160242.

[69] A. Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 2008. 240 pp. ISBN: 0-321-60647-7.

[70] K. Kravari and N. Bassiliades. 'A Survey of Agent Platforms'. In: *Journal of Artificial Societies and Social Simulation* 18.1 (2015), pp. 1–18. ISSN: 14607425. DOI: 10.18564/jasss.2661.

[71] V. Krishnan and S. Martínez. 'Distributed Control for Spatial Self-Organization of Multi-Agent Swarms'. In: (8th May 2017). arXiv: 1705.03109 [math].

[72] M. A. Laouadi, F. Mokhati and H. Seridi. 'A Novel Organizational Model for Real Time MAS: Towards a Formal Specification'. In: *Intelligent Systems for Science and Information*. Ed. by L. Chen, S. Kapoor and R. Bhatia. Studies in Computational Intelligence 542 542. Cham, CH: Springer, 2014. Chap. 10, pp. 171–180. ISBN: 978-3-319-04702-7. DOI: 10.1007/978-3-319-04702-7_10.

[73] K. M. Lhaksmana, Y. Murakami and T. Ishida. 'Role-Based Modeling for Designing Agent Behavior in Self-Organizing Multi-Agent Systems'. In: *International Journal of Software Engineering and Knowledge Engineering* 28.01 (2018), pp. 79–96. ISSN: 0218-1940. DOI: 10.1142/S0218194018500043.

[74] F. J. M. Lizán and C. R. Maestre. 'Intelligent Buildings: Foundation for Intelligent Physical Agents'. In: *International Journal of Engineering Research and Applications* 7.5 (May 2017), pp. 21–25. ISSN: 22489622. DOI: 10.9790/9622-0705022125.

[75] M. Lopez et al. 'Building a Chemical Ontology Using Methontology and the Ontology Design Environment'. In: *IEEE Intelligent Systems* 14.1 (1999), pp. 37–46. ISSN: 1094-7167. DOI: 10.1109/5254.747904.

[76] M. Luck and R. Aylett. 'Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments'. In: *Applied Artificial Intelligence* 14.1 (2000), pp. 3–32. ISSN: 0883-9514, 1087-6545. DOI: 10.1080/088395100117142.

[77] C. N. Madu. 'Simulation in Manufacturing: A Regression Metamodel Approach'. In: *Computers & Industrial Engineering* 18.3 (1990), pp. 381–389. ISSN: 03608352. DOI: 10.1016/0360-8352(90)90060-Y.

[78] M. A. Mahmoud et al. 'A Review of Norms and Normative Multiagent Systems'. In: *The Scientific World Journal* 2014 (2014), pp. 1–23. ISSN: 2356-6140. DOI: 10.1155/2014/684587.

[79] M. Maleković. 'Multi-Agent Systems: Incorporating Knowledge and Time'. In: *Journal of Information and Organizational Sciences* 22.2 (1998), pp. 97–105.

[80] M. Maleković and M. Schatten. *Teorija i primjena baza podataka*. 1st ed. Varaždin, HR: Faculty of Organization and Informatics, University of Zagreb, 2017. 427 pp. ISBN: 978-953-6071-62-3.

[81] J.-J. C. Meyer and R. J. Wieringa, eds. *Deontic Logic in Computer Science: Normative System Specification*. Chichester, UK: John Wiley and Sons Ltd., 1993. ISBN: 0-471-93743-6.

[82] J. A. Miller, A. P. Sheth and K. J. Kochut. 'Perspectives in Modeling: Simulation, Database, and Workflow'. In: *Conceptual Modeling: Current Issues and Future Directions*. Ed. by P. P. Chen et al. Lecture Notes in Computer Science 1565 1565. Berlin, Heidelberg: Springer, 1999, pp. 154–167. ISBN: 978-3-540-48854-5. DOI: 10.1007/3-540-48854-5_13.

[83] E. Missaoui et al. 'A Normative Model for Holonic Multi-Agent Systems'. In: *IEEE ICTAI-17*. Boston, USA: IEEE, 2017.

[84] S. Mitchell et al. *The Internet of Everything for Cities*. Cisco, 2013, pp. 1–21.

[85] W. Muhanna and R. Pick. 'Meta-Modeling Concepts and Tools for Model Management: A Systems Approach'. In: *Management Science* 40.9 (1994), pp. 1093–1123. ISSN: 0025-1909. DOI: 10.1287/mnsc.40.9.1093.

[86] M. A. Musen. 'The Protégé Project: A Look Back and a Look Forward'. In: *AI Matters* 1.4 (2015), pp. 4–12. ISSN: 23723483. DOI: 10.1145/2757001.2757003.

[87] D. Nadler. *Organizational Architecture : Designs for Changing Organizations*. Ed. by D. A. Nadler, M. S. Gerstein and R. B. Shaw. San Francisco, USA: Jossey-Bass, 1992. 284 pp. ISBN: 1-55542-443-0.

[88] M. Nagl. 'Formal Languages of Labelled Graphs'. In: *Computing* 16.1-2 (1976), pp. 113–137. ISSN: 0010485X. DOI: 10.1007/BF02241984.

[89] M. Nagl. *Graph-Grammatiken*. Wiesbaden, DE: Vieweg+Teubner Verlag, 1979. 378 pp. ISBN: 978-3-528-03338-5. DOI: 10.1007/978-3-663-01443-0.

[90] M. Navarro, V. Julian and V. Botti. 'jTRASTO: A Development Toolkit for Real-Time Multi-Agent Systems'. In: *Multi-Agent Systems and Applications V*. International Central and Eastern European Conference on Multi-Agent Systems. Ed. by H.-D. Burkhard et al. Lecture Notes in Computer Science 4696. Berlin, Heidelberg: Springer, 2007. Chap. 39, pp. 325–327. DOI: 10.1007/978-3-540-75254-7_39.

[91] M. Navarro et al. 'Towards Real-Time Argumentation'. In: *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 4.4 (2015), pp. 35–58. DOI: 10.14201/ADCAIJ2015443558.

[92] B. Okreša Đurić. 'A Novel Approach to Modelling Distributed Systems: Using Large-Scale Multi-Agent Systems'. In: *Software Project Management for Distributed Computing*. Ed. by Z. Mahmood. 1st ed. Springer International Publishing AG, 2017. Chap. 10, pp. 229–254. ISBN: 978-3-319-54325-3. DOI: 10.1007/978-3-319-54325-3_10.

[93] B. Okreša Đurić. 'Organisational Metamodel for Large-Scale Multi-Agent Systems: First Steps Towards Modelling Organisation Dynamics'. In: *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 6.3 (2017), p. 17. ISSN: 2255-2863. DOI: 10.14201/ADCAIJ2017631727.

[94] B. Okreša Đurić. 'Organizational Metamodel for Large-Scale Multi-Agent Systems'. In: *Trends in Practical Applications of Scalable Multi- Agent Systems, the PAAMS Collection*. Ed. by F. de la Prieta et al. Advances in Intelligent Systems and Computing 473. Seville, ES: Springer International Publishing, 2016. Chap. 8, pp. 387–390. ISBN: 978-3-319-40158-4. DOI: 10.1007/978-3-319-40159-1_36.

[95] B. Okreša Đurić. 'Semantic Modeling of Business Rules'. MA thesis. University of Zagreb, 2013. 73 pp.

[96] B. Okreša Đurić. 'Towards Modelling Organisational Dynamics for Large-Scale Multiagent Systems'. In: *Trends in Cyber-Physical Multi-Agent Systems. The PAAMS Collection - 15th International Conference, PAAMS 2017*. Ed. by F. De la Prieta et al. Advances in Intelligent Systems and Computing 619. Cham: Springer International Publishing, 16th July 2017, pp. 245–248. ISBN: 978-3-319-61578-3. DOI: 10.1007/978-3-319-61578-3_28.

[97] B. Okreša Đurić and M. Konecki. 'Modeling MMORPG Players' Behaviour'. In: *Central European Conference on Information and Intelligent Systems*. Ed. by T. Hunjak, V. Kirinić and M. Konecki. Varaždin, HR: University of Zagreb, Faculty of Organization and Informatics Varaždin, 2015, pp. 177–184.

[98] B. Okreša Đurić and M. Konecki. 'Specific OWL-Based RPG Ontology'. In: *Central European Conference on Information and Intelligent Systems*. Ed. by T. Hunjak, V. Kirinić and M. Konecki. Varaždin, HR: University of Zagreb, Faculty of Organization and Informatics Varaždin, 2015, pp. 185–190.

[99] B. Okreša Đurić and M. Maleković. 'How to Manage Knowledge With Domain Specific and General Conceptual Modelling Examples'. In: *Proceedings of the 19th European Conference on Knowledge Management*. European Conference on Knowledge Management. Ed. by E. Bolisani, E. Di Maria and E. Scarso. Vol. 2. Reading,

UK: Academic Conferences and Publishing International Limited, 6th Sept. 2018, pp. 615–622. ISBN: 978-1-911218-95-1.

[100] B. Okreša Đurić and M. Maleković. 'Knowledge Management and Conceptual Modelling Towards Better Business Results'. In: *Proceedings of the ENTRENOVA - ENTerprise REsearch InNOVAtion Conference*. ENTerprise REsearch InNOVAtion Conference. Ed. by M. Milković et al. Split, HR: Udruga za promicanje inovacija i istraživanja u ekonomiji "IRINET", Zagreb, Croatia, 2018, pp. 239–245.

[101] B. Okreša Đurić and M. Schatten. 'Defining Ontology Combining Concepts of Massive Multi-Player Online Role Playing Games and Organization of Large-Scale Multi-Agent Systems'. In: *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Opatija, HR: IEEE, 2016, pp. 1330–1335. ISBN: 978-953-233-086-1. DOI: 10.1109/MIPRO.2016.7522346.

[102] B. Okreša Đurić and M. Schatten. 'Modeling Multiagent Knowledge Systems Based on Implicit Culture'. In: *Central European Conference on Information and Intelligent Systems*. Ed. by T. Hunjak, S. Lovreňcić and I. Tomĭcić. Varaždin, HR: University of Zagreb, Faculty of Organization and Informatics Varaždin, 2012, pp. 57–61.

[103] B. Okreša Đurić, I. Tomičić and M. Schatten. 'Towards Agent-Based Simulation of Emerging and Large-Scale Social Networks. Examples of the Migrant Crisis and MMORPGs'. In: *European Quarterly of Political Attitudes and Mentalities EQPAM* 5.4 (2016), pp. 1–19.

[104] B. Okreša urić et al. 'MAMbO5: A New Ontology Approach for Modelling and Managing Intelligent Virtual Environments Based on Multi-Agent Systems'. In: *Journal of Ambient Intelligence and Humanized Computing* (12th Oct. 2018). ISSN: 1868-5137, 1868-5145. DOI: 10.1007/s12652-018-1089-4.

[105] A. Olivé. *Conceptual Modeling of Information Systems*. Berlin, Heidelberg: Springer, 2007. 471 pp. ISBN: 978-3-540-39389-4. DOI: 10.1007/978-3-540-39390-0.

[106] A. Omicini, A. Ricci and M. Viroli. 'Artifacts in the A&A Meta-Model for Multi-Agent Systems'. In: *Autonomous Agents and Multi-Agent Systems* 17.3 (2008), pp. 432–456. ISSN: 13872532. DOI: 10.1007/s10458-008-9053-x.

[107] M. A. Paredes-Valverde et al. 'ONLI: An Ontology-Based System for Querying DBpedia Using Natural Language Paradigm'. In: *Expert Systems with Applications* 42.12 (July 2015), pp. 5163–5176. ISSN: 09574174. DOI: 10.1016/j.eswa.2015.02.034.

[108] J. Parkkila et al. *The Video Game Ontology*. 2014. URL: http://vocab.linkeddata.es/vgo/ (visited on 29/07/2018).

[109] E. Posse, J. de Lara and H. Vangheluwe. 'Processing Causal Block Diagrams with Graphgrammars in Atom3'. In: *European Joint Conference on Theory and Practice of Software (ETAPS), Workshop on Applied Graph Transformation (AGT)*. 2002, pp. 23–34.

[110] S. F. Railsback and V. Grimm. *Agent-Based and Invidual-Based Modeling*. 2012. 329 pp. ISBN: 978-0-691-13674-5.

[111] J. A. Rincon, C. Carrascosa and E. Garcia. 'Developing Intelligent Virtual Environments Using MAM5 Meta-Model'. In: *Advances in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*. Ed. by Y. Demazeau et al. Lecture Notes in Computer Science 8473. Cham, CH: Springer, 2014, pp. 379–382. ISBN: 9783319075501. DOI: 10.1007/978-3-319-07551-8_43.

[112] J. A. Rincon et al. 'Developing Adaptive Agents Situated in Intelligent Virtual Environments'. In: *Hybrid Artificial Intelligence Systems*. International Conference on Hybrid Artificial Intelligent Systems. Vol. 8480 LNAI. Lecture Notes in Computer Science 8480. Cham, CH: Springer, 2014, pp. 98–109. ISBN: 9783319076164. DOI: 10.1007/978-3-319-07617-1_9.

[113] S. Rodriguez et al. 'Holonic Multi-Agent Systems'. In: *Self-Organising Software: From Natural to Artificial Adaptation*. Ed. by G. Di Marzo Serugendo, M.-P. Gleizes and A. Karageorgos. Natural Computing Series. Berlin, Heidelberg: Springer, 2011, pp. 251–279. ISBN: 978-3-642-17347-9. DOI: 10.1007/978-3-642-17348-6_11.

[114] M. Roman, I. Sandu and S. C. Buraga. 'OWL-Based Modeling of RPG Games'. In: *Studia Universitatis Babes-Bolyai* 56.3 (2011), pp. 83–90.

[115] G. Rozenberg, ed. *Handbook of Graph Grammars and Computing by Graph Transformation*. River Edge, NJ, USA: World Scientific Publishing, 1997. 572 pp. ISBN: 981-238-472-3.

[116] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Ed. by S. Russell and P. Norvig. 3rd ed. Prentice Hall Series in Artificial Intelligence. New Jersey, USA: Prentice Hall, 2010. 1132 pp. ISBN: 978-0-13-604259-4.

[117] R. G. Sargent. 'Research Issues in Metamodeling'. In: *Proceedings of the 1991 Winter Simulation Conference*. Winter Simulation Conference. Ed. by B. L. Nelson, W. D. Kelton and G. M. Clark. Phoenix, AR, USA: IEEE Computer Society, 1991, pp. 888–893. ISBN: 0-7803-0181-1.

[118] M. Schatten. 'Organizational Architectures for Large-Scale Multi-Agent Systems' Development: An Initial Ontology'. In: *Advances in Intelligent Systems and Computing* 290 (2014). Ed. by S. Omatu et al., pp. 261–268. DOI: 10.1007/978-3-319-07593-8_31.

[119] M. Schatten. ‘Reorganization in Multi-Agent Architectures: An Active Graph Grammar Approach’. In: *Business Systems Research* 4.1 (2013), pp. 14–20. ISSN: 1847-9375. DOI: 10.2478/bsrj-2013-0002.

[120] M. Schatten and B. Okreša Đurić. ‘A Social Network Analysis of a Massively Multi-Player On-Line Role Playing Game’. In: *Proceedings of the 4th International Conference on Modeling and Simulation*. Ed. by B. Kang. Jeju Island, Korea: IEEE, 2015, pp. 37–42. ISBN: 978-1-4673-9828-2. DOI: 10.1109/MAS.2015.19.

[121] M. Schatten and B. Okreša Đurić. ‘Social Networks in "The Mana World" - an Analysis of Social Ties in an Open Source MMORPG’. In: *International Journal of Multimedia and Ubiquitous Engineering* 11.3 (2016), pp. 257–272. DOI: 10.14257/ijmue.2016.11.3.25.

[122] M. Schatten, J. Ševa and I. Tomičić. ‘A Roadmap for Scalable Agent Organizations in the Internet of Everything’. In: *Journal of Systems and Software* 115 (2016), pp. 31–41. ISSN: 01641212. DOI: 10.1016/j.jss.2016.01.022.

[123] M. Schatten, I. Tomičić and B. Okreša Đurić. ‘Multi-Agent Modeling Methods for Massivley Multi-Player On-Line Role-Playing Games’. In: *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Ed. by P. Biljanović. Opatija, HR: IEEE, 2015, pp. 1256–1261. ISBN: 978-953-233-082-3. DOI: 10.1109/MIPRO.2015.7160468.

[124] M. Schatten et al. ‘Agents as Bots – An Initial Attempt Towards Model-Driven MMORPG Gameplay’. In: *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection*. Ed. by Y. Demazeau et al. Lecture Notes in Artificial Intelligence 10349. Cham, Switzerland: Springer International Publishing, 2017, pp. 246–258. ISBN: 978-3-319-59930-4. DOI: 10.1007/978-3-319-59930-4_20.

[125] M. Schatten et al. ‘Automated MMORPG Testing – An Agent-Based Approach’. In: *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection*. Ed. by Y. Demazeau et al. Lecture Notes in Artificial Intelligence 10349. Cham, Switzerland: Springer International Publishing, 2017, pp. 359–363. ISBN: 978-3-319-59930-4. DOI: 10.1007/978-3-319-59930-4_38.

[126] M. Schatten et al. ‘Towards a Formal Conceptualization of Organizational Design Techniques for Large Scale Multi Agent Systems’. In: *Procedia Technology* 15 (2014), pp. 576–585. ISSN: 22120173. DOI: 10.1016/j.protcy.2014.09.018.

[127] M. Schatten et al. ‘Towards an Agent-Based Automated Testing Environment for Massively Multi-Player Role Playing Games’. In: *MIPRO 2017 40th Jubilee International Convention Proceedings* (2017), pp. 1361–1366. DOI: 10.23919/MIPRO.2017.7973597.

[128] G. Schreiber. 'Knowledge Engineering'. In: *Handbook of Knowledge Representation*. Ed. by F. van Harmelen, V. Lifschitz and B. Porter. Foundations of Artificial Intelligence 3. Elsevier, 2008. Chap. 25, pp. 929–946. ISBN: 978-0-444-52211-5. DOI: 10.1016/S1574-6526(07)03025-8.

[129] B. Sharp, A. Atkins and H. Kothari. 'An Ontology Based Multi-Agent System to Support HABIO Outsourcing Framework'. In: *Expert Systems with Applications* 38.6 (June 2011), pp. 6949–6956. ISSN: 09574174. DOI: 10.1016/j.eswa.2010.12.020.

[130] A. Sharpanskykh. 'Modeling of Agents in Organizational Context'. In: *Multi-Agent Systems and Applications V*. International Central and Eastern European Conference on Multi-Agent Systems. Ed. by H.-D. Burkhard et al. Lecture Notes in Computer Science 4696. Berlin, Heidelberg: Springer, 2007. Chap. 20, pp. 193–203. DOI: 10.1007/978-3-540-75254-7_20.

[131] B. Smith. *Ontology and Information Systems*. 2002.

[132] L. S. Sterling and K. Taveter. *The Art of Agent-Oriented Modeling*. Ed. by R. C. Arkin. London, UK: The MIT Press, 2009. 367 pp. ISBN: 978-0-262-26004-6.

[133] G. Sukthankar and J. A. Rodriguez-Aguilar, eds. *Autonomous Agents and Multiagent Systems*. Lecture Notes in Computer Science 10642 10642. Cham, CH: Springer, 2017. ISBN: 978-3-319-71681-7. DOI: 10.1007/978-3-319-71682-4.

[134] B. Thalheim. 'The Theory of Conceptual Models, the Theory of Conceptual Modelling and Foundations of Conceptual Modelling'. In: *Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges*. Ed. by D. W. Embley and B. Thalheim. Berlin, Heidelberg: Springer, 2011. Chap. 17, pp. 543–577. DOI: 10.1007/978-3-642-15865-0_17.

[135] I. Tomičić. 'Agent-Based Framework for Modelling and Simulation of Resource Management in Smart Self-Sustainable Human Settlements'. Doctoral thesis. Varaždin, HR: University of Zagreb, 2016. 250 pp.

[136] I. Tomičić, B. Okreša Đurić and M. Schatten. 'Modeling Smart Self-Sustainable Cities as Large-Scale Agent Organizations in the IoT Environment'. In: *Smart Cities: Development and Governance Frameworks*. Ed. by Z. Mahmood. Computer Communications and Networks. Cham, CH: Springer, 2018, pp. 3–23. ISBN: 978-3-319-76668-3. DOI: 10.1007/978-3-319-76669-0_1.

[137] I. Tomičić and M. Schatten. 'Agent-Based Framework for Modeling and Simulation of Resources in Self-Sustainable Human Settlements: A Case Study on Water Management in an Eco-Village Community in Croatia'. In: *International Journal of Sustainable Development & World Ecology* 23.6 (2016), pp. 504–513. ISSN: 1350-4509. DOI: 10.1080/13504509.2016.1153527.

[138] I. Tomičić et al. 'Self-Sustainable Agent Organizations in Massively Multi-Player On-Line Role-Playing Games – A Conceptual Framework'. In: *Central European Conference on Information and Intelligent Systems*. Central European Conference on Information and Intelligent Systems. Ed. by T. Hunjak, V. Kirinić and M. Konecki. Varaždin, HR: University of Zagreb, Faculty of Organization and Informatics Varaždin, 2016, pp. 213–217.

[139] A. Tsarev and P. Skobelev. 'Multi-Agent Supply Scheduling System Prototype for Energy Production and Distribution'. In: *Advances in Practical Applications of Scalable Multi-Agent Systems. The PAAMS Collection*. International Conference on Practical Applications of Scalable Multi-Agent Systems. Ed. by Y. Demazeau et al. Lecture Notes in Computer Science 9662. Cham, CH: Springer, 2016, pp. 290–293. ISBN: 978-3-319-39324-7. DOI: 10.1007/978-3-319-39324-7_33.

[140] M. Uschold and M. Gruninger. 'Ontologies: Principles, Methods and Applications'. In: *The Knowledge Engineering Review* 11.2 (1996), pp. 93–136. DOI: 10.1017/S0269888900007797.

[141] H. Van Dyke Parunak and J. Odell. 'Representing Social Structures in UML'. In: *Proceedings of the Fifth International Conference on Autonomous Agents*. International Conference on Autonomous Agents. New York, NY, USA: ACM Press, 2001, pp. 100–101. ISBN: 1-58113-326-X. DOI: 10.1145/375735.376008.

[142] O. Vermesan et al. *Internet of Things Strategic Research Roadmap*. Strategic Research Agenda. European Research Cluster on the Internet of Things, 2009, pp. 9–52.

[143] D. Villatoro. 'Self-Organization in Decentralized Agent Societies Through Social Norms'. In: *The 10th International Conference on Autonomous Agents and Multiagent Systems*. International Conference on Autonomous Agents and Multiagent Systems. Vol. 3. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 1373–1374. ISBN: 978-0-9826571-7-1.

[144] P. Vlacheas et al. 'Enabling Smart Cities through a Cognitive Management Framework for the Internet of Things'. In: *IEEE Communications Magazine* 51.6 (2013), pp. 102–111. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6525602.

[145] W3C. *The Organization Ontology*. 16th Jan. 2014. URL: https://www.w3.org/TR/vocab-org/ (visited on 22/02/2016).

[146] W3C OWL Working Group. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. 11th Dec. 2012. URL: http://www.w3.org/TR/owl2-overview/ (visited on 14/05/2015).

[147] W3C OWL Working Group. *OWL 2 Web Ontology Language Primer (Second Edition)*. 11th Dec. 2012. URL: https://www.w3.org/TR/2012/REC-owl2-primer-20121211/ (visited on 06/06/2018).

[148] Y. Wand. 'Ontology as a Foundation for Meta-Modelling and Method Engineering'. In: *Information and Software Technology* 38.4 (1996), pp. 281–287. ISSN: 0950-5849. DOI: 10.1016/0950-5849(95)01052-1.

[149] Y. Wand and C. Woo. 'Object-Oriented Analysis - Is It Really That Simple?' In: *Proceedings of the 3rd Workshop on Information Technologies and Systems*. Orlando, FL, USA, 1993, pp. 186–195.

[150] W. Wang et al. 'Knowledge Representation in the Internet of Things: Semantic Modelling and Its Applications'. In: *Automatika* 54.4 (2013), pp. 388–400. DOI: 10.7305/automatika.54-4.414.

[151] G. Weiss, ed. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. 3rd ed. London, UK: The MIT Press, 2001. ISBN: 0-262-23203-0.

[152] E. Welbourne et al. 'Building the Internet of Things Using RFID'. In: *IEEE Internet Computing* 13.3 (2009), pp. 48–55. ISSN: 1089-7801. DOI: 10.1109/MIC.2009.52.

[153] M. P. Wellman. 'Putting the Agent in Agent-Based Modeling'. In: *Autonomous Agents and Multi-Agent Systems* 30.6 (2016), pp. 1175–1189. ISSN: 1387-2532. DOI: 10.1007/s10458-016-9336-6.

[154] D. Weyns, R. Haesevoets and A. Helleboogh. 'The MACODO Organization Model for Context-Driven Dynamic Agent Organizations'. In: *ACM Transactions on Autonomous and Adaptive Systems* 5.4 (2010), 16:1–16:29. ISSN: 15564665. DOI: 10.1145/1867713.1867717.

[155] D. Weyns et al. 'The MACODO Middleware for Context-Driven Dynamic Agent Organizations'. In: *ACM Transactions on Autonomous and Adaptive Systems* 5.1 (2010), 3:1–3:28. ISSN: 1556-4665. DOI: 10.1145/1671948.1671951.

[156] R. J. Wilson. *Introduction to Graph Theory*. 4th ed. Essex, UK: Addison Wesley Longman Limited, 1996. ISBN: 0-582-24993-7.

[157] M. Žugaj. *Znanstvena istraživanja u društvenim znanostima i nastanak znanstvenog djela*. Varaždinske Toplice, HR: Tonimir, 2007. 215 pp.

Appendices

Appendix A

METHONTOLOGY

A.1 Data Dictionary

Table A.1: *Acquisition* data dictionary entry

Concept name	Acquisition
Definition	An acquisition is the purchase of all or a portion of a corporate asset or target company[1].
Description	An acquisition is, in economical terms, described as, in layman's terms, one company buying another. This is usually done using stocks - the buyer buys most of the target company's ownership stakes to assume control of it[2]. Reasons for performing acquisitions are numerous, including to achieve economies of scale, greater market share, increased synergy, cost reductions, or new niche offerings.

Table A.2: *Action* data dictionary entry

Concept name	Action (C)
Synonyms	Activity, Behaviour, Agent Action
Definition	An action is the building block of agents' activities.
Description	An action is esentially an agent's response to tasks. Whereby tasks are created to be met or reached, an action is the atomic concept for achieving tasks. In the context of this document, an action is the building block of a process, and agents' ability to act towards its environment in general. Every action can be used to fulfill at least one task.
Instance/s	Attack, PickItem, GoToLocation, BrewPotion, MakeItem

Table A.3: *Agent* data dictionary entry

Concept name	Agent (A)
Synonyms	Organisational Individual
Definition	A piece of software that can act upon its environment and perceive it.
Description	An agent in the context of this document is a piece of software that can interact with its environment, act upon it, and, in case of an intelligent agent, reason upon their accessible knowledge. Indeed, an agent is *anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.* [116] In the organisational context of this document, a software agent is essentially a model of a real-life person.

Table A.4: *Artefact* data dictionary entry

Concept name	Artefact
Definition	An artefact is an otherwise unclassified element of an organisation system.
Description	An artefact is, as of yet, a somewhat undefined concept, in the context of specifying its domain. Essentially, an artefact can be anything that is not classified using the other classes of this ontology. Furthermore, an artefact can be phisically representative (e.g. a chair), or an unphisical concept (e.g. knowledge). Artefacts therefore represent various concepts that the agents can interact with, or that affect the given environment or the given system, i.e. objects forming the environment.

Table A.5: *Criteria of Organising* data dictionary entry

Concept name	Criteria of Organising
Definition	
Description	This concept comes from the OOVASIS ontology [118, 126] where it represents varius criteria of organising agents within an organisation. One of the criteria is ..., another ... . Therefore, this concept determines what are the grounds for creating the given organisation in the first place, and governs the decision flow in the context of deciding which organisational features (starting from architecture) are most suitable for the given criteria of organising.

Table A.6: *Design Factor* data dictionary entry

Concept name	Design Factor
Definition	A design factor is an internal or an external factor with significant influence on the design of an organisation.
Description	Everything that influences the design of an organisation on a non-neglectable level is considered a design factor. Design factors can be internal and external, relative to the given organisation. [126].
Instance/s	development of science and technology, human resources, market, size of organisation, strategy, etc.

Table A.7: *Design Method* data dictionary entry

Concept name	Design Method
Synonyms	Organisational Design Method
Definition	A design method is a common organisational design practice dealing with various aspects of organisational architecture.
Description	Every design method addresses a number of aspects of organisational architecture. A design method is esentially a common organisational design practice. [126]
Instance/s	business process reingeneering, kaizen, six sigma, lean management, knowledge management, etc.

Table A.8: *Goal* data dictionary entry

Concept name	Goal (G)
Definition	A goal is a result towards which effort is directed - an end to be met.
Description	A goal is broadly defined as a result or achievement towards which effort is directed[3]. In the context of this document, a goal is a form of an objective. A goal is an end to be met or reached, and can consist of several sub-goals.

Table A.9: *Heterarchical Organisational Structure* data dictionary entry

Concept name	Heterarchical Organisational Structure
Definition	Heterarchical organisational structure is an organisational structude without a single clearly defined pyramid-like structure.
Description	When there is no single clear pyramid-like line of control in an organisation, the given organisation can be described as having a heterarchical organisational structure. As opposed to hierarchical organisational structure, heterarchical organisational structure can be visualised as an oriented forest [4], or essentially using a network-based visualisation [126].
Instance/s	fishnet structure, Hollywood structure, spaghetti structure, etc.

Table A.10: *Hierarchical Organisational Structure* data dictionary entry

Concept name	Hierarchical Organisational Structure
Definition	Hierarchical organisational structure is an organisational structude with a single clearly defined pyramid-like structure.
Description	In contrast to the heterarchical organisational structure, hierarchical organisational structure can be identified by its basic pyramid-like form fostering hierarchical relations between organisation units. Such an organisational structure can be visualised using an oriented tree [4].
Instance/s	functional structure, project-oriented structure, matrix, etc.

Table A.11: *Human Immersed Agent* data dictionary entry

Concept name	Human Immersed Agent
Definition	Real-world agents that are represented in a IVE using their wearable tecchnology gadgets.
Description	Humans can be represented within a IVE and be available for interaction with the digital agents within the environment using digital aids, most prominently featured as wearable technology items, such as smartwatches and similar. Such agents are dubbed human immersed agents, since they are real-life people represented in the digital world using their attached piece of wearable discreet equipment.

Table A.12: *Hybrid Organisational Structure* data dictionary entry

Concept name	Hybrid Organisational Structure
Definition	Having mixed aspects of both heterarchical and hierarchical organisational structures, a hybrid organisational structure is a blend of the two.
Description	Having mixed aspects of both heterarchical and hierarchical organisational structures, a hybrid organisational structure is a blend of the two.
Instance/s	academic structure, front-back structure, inverted structure, etc.

Table A.13: *Inhabitant Agent* data dictionary entry

Concept name	Inhabitant Agent
Definition	Every agent that is can be represented as phisically present in an IVE is considered an inhabitant agent.
Description	Agents that can be phisically represented within a IVE are called inhabitant agents. These agents can be of artificial or real-world nature. Usually various IVE artefacts exist within the IVE that represent various inhabitant agents [112]. It could be said that these agents have their habitats within their respective IVEs.
Instance/s	Archmage, Hermit, Sorfina, mali_agent13

Table A.14: *Intelligent Virtual Environment* data dictionary entry

Concept name	Intelligent Virtual Environment (IVE)
Definition	An intelligent virtual environment is a virtual environment that simulates the real world, and is populated by autonomous intelligent entities. [111]
Description	Intelligent virtual environments are researched as an area on the intersection of two aspects pertaining to the concept of artificial intelligence, if only but marginally: intelligent tools and techniques that are embodied in autonomous agents (real-life and digital alike), and effective ways of representing them, along with various means of achieving different kinds of interaction amongst them [111, 76]. In other words, a IVE is a concepte that represents a virtual environment whose main goal is simulating a segment of the real world, populated by artificial autonomous entities (agents). [111]
Instance/s	modified version of The Mana World

Table A.15: *IVE Law* data dictionary entry

Concept name	IVE Law
Definition	A IVE law is a norm that is valid only within a specified physical space (a IVE workspace).
Description	A special kind of a norm, an IVE law is a norm that is constrained by its applicability to a specific physical space, i.e. a specific IVE workspace. Being applicable to only a restricted area means that every IVE law is valid only within the bounds of the given area (a IVE workspace), and never outside of that specified space. This kind of a norm is the key constraint of the concept of a situated organisational unit.
Instance/s	When a character is located on a map with at least 75% of tiles of type Frozen, they are more suspectible to Damage of type Ice.

Table A.16: *IVE Workspace* data dictionary entry

Concept name	IVE Workspace
Definition	
Description	Complimentary to the concept of a workspace, a IVE workspace represents a physical location, or a physically describable location.

Table A.17: *Knowledge Artefact* data dictionary entry

Concept name	Knowledge Artefact (KnArt)
Definition	Knowledge artefact is a piece of knowledge of an agent or an organisation.
Description	A knowledge artefact is a piece of knowledge, or a set of knowledge terms available to agents within the system or within the IVE. Depending on the wanted level of abstraction, a knowledge artefact may represent a database containing various pieces of knowledge accessible by sets of agents, or individual pieces of knowledge. In the terms of rather undefined artefact class, knowledge artefacts are yet to be perfected in the context of knowledge representation and their suitability for representing knowledge of a IVE or a MAS.
Instance/s	organisational culture rulebook
Attributes	isAccessibleTo

Table A.18: *Manual* data dictionary entry

Concept name	Manual
Definition	
Description	A manual defines the interface between individual agents and artefacts of a IVE. Including such a concept in the description of a IVE domain helps reduce unnecessary clutter in the context of setting ground-rules of how to use an artefact up front. The agents therefore immediately learn of the possibilities and applications of a given artefact without the need for exploring its possible uses.

Table A.19: *Merger* data dictionary entry

Concept name	Merger
Definition	A merger is the process of organisational integration.
Description	In standard economical terms, a merger is a combination of more than one company by the transfer of the properties to one surviving company[4]. In the context of this document, merger can simply be regarded as an organisational integration.

Table A.20: *Norm* data dictionary entry

Concept name	Norm
Definition	*Norms are informal rules that are socially enforced.* [78]
Description	Norms in general are not very different from the definition of a rule, their more generic counterpart. Used in a context of a population of a community, be it a natural or an artificial one, norms are expressions of desirable behaviour generally understood as rules indicating actions that are expected to be pursued. Norms are basically divided in three types: obligatory, prohibitive, and permissive. In the context of normative MASs though, there are three different terms associated with norms: conventions, social norms, and social laws [78, 143], and two categories [26]: conventions and essential norms.
Instance/s	Formal Dress Code, The Dragon Egg item is usable for at most 23 hours after being laid.

Table A.21: *Normative System* data dictionary entry

Concept name	Normative System
Definition	*Systems in the behaviour of which norms play a role and which need normative concepts in order to be described or specified [...]* [14, 81]
Description	A normative system is a system built on norms and their enfonrcement upon the system, or system's definition of architecture based on the said norms. In the context of computer science, a normative system is described as a system whose behaviour is influenced by norms, and whose description or specification depencds on using normative concepts [14, 81].

Table A.22: *Objective* data dictionary entry

Concept name	Objective (O)
Definition	An objective is a high-level goal the be met, suitable for the context of strategic planning.
Description	An objective is more general than a goal, although their definitions are rather similar. Fulfilling several goals can lead an organisational unit towards fulfilling a set objective. Thus, an objective is more suitable in the context of strategic planning, while a goal is more suitably used in the context of short-term planning.
Instance/s	LearnSpell, FindDragonEgg, Brew Hatching Potion
Attributes	triggers, hasCriteriaOfOrganizing, isAchievedBy

Table A.23: *Observable Property* data dictionary entry

Concept name	Observable Property
Definition	An observable property is a peroperty of an artefact that can be observed by agents in the same IVE.
Description	This is a property of an artefact located in a IVE that is observable by other agents located within the same IVE. These are tighly connected to the concept of observable events, and can be influenced upon by an operation.

Table A.24: *Organisation* data dictionary entry

Concept name	Organisation
Definition	An organisation is generally a group of agents structured according to a set criteria, with the basic goal of overcoming limitations of individual agency and achieving an organisation goal.
Description	An apt definition is given in [22] where an organisation is defined using several characteristics, including large-scale problem solving technology, composition of multiple agents, systems of goal-directed activities, etc. Furthermore, an essential benefit of organisations is identified in overcoming limitations of individual agency, especially cognitive, physical, temporal, and institutional.

Table A.25: *Organisational Architecture* data dictionary entry

Concept name	Organisational Architecture
Definition	In the context of this document, organisational architecture is the superclass for all the organisation-related concepts that deal with more than one aspect of organisational architecture.
Description	All those concepts that deal with more than one aspect of organisational architecture, i.e. are not specialised as for example concepts that describe organisational structure only, are classified as belonging to the organisational architecture concept. [126] therefore identifies 15 such concepts.
Instance/s	Shamrock organisation, strategic organisation, information-based organisation, learning organisation, open organisation, etc.

Table A.26: *Organisational Change* data dictionary entry

Concept name	Organisational Change
Synonyms	Organisational Dynamics
Definition	
Description	The concept of organisational change is closely tied to the intension of the concept of orgnaisational dynamics, since both concepts describe change to the established agent organisations. A change in the context of organisational change definition can be influenced by an organisational design method, yet unmistakingly it affects the organisational architecture of the given organisation. A change as defined here can adhere to one of the identified types of change (e.g. structural, cultural, strategic, etc.), can be attributed an impact of change, reason why the change started, and a key influence area (e.g. organisational memory) [126].

Table A.27: *Organisational Culture* data dictionary entry

Concept name	Organisational Culture
Definition	*Organizational culture defines important intangible aspects of an organization including knowledge, social norms, reward systems, language and similar.* [122, 118]
Description	The concept of organisational culture encompasses all the intangible aspects of an organisation, such as knowledge, various types of norms, a system of rewards, languages used in the organisation, etc. Organisational culture is therefore a concept that is mostly based in the organisational units, i.e. in the individual agents forming the organisation, and is thus the most fuzzy concept of all the perspectives of an organisation. [122, 126] provide a quick overview of various conceptualisations of organisational architecture, where it is visible that organisational culture is an important part of an organisation.

Table A.28: *Organisational Environment* data dictionary entry

Concept name	Organisational Environment
Definition	Organisational environment are all the external factors that have the capacity to influence an organisation.
Description	The concept of organisational environment encompasses all the concepts that represent factors external to an organisation that have a potential to influence the given organisation, such as external organisations or individuals, or external events. Main concerns when organisational environment is considered are directed towards identifying constraints imposed on the given organisation by the environment, and demands of the environment towards the given organisation. [122]

Table A.29: *Organisational Knowledge Network* data dictionary entry

Concept name	Organisational Knowledge Network
Definition	Organisational knowledge network is a network created by interconnected pieces of organisational knowledge.
Description	A network connecting all the pieces of organisational knowledge is considered to build an organisational knowledge network that effectively collects and intertwines all the knowledge of an organisation, thus fostering knowledge sharing and reuse amongst the organisational units of the given organisation, i.e. ultimately individual agents.

Table A.30: *Organisational Structure* data dictionary entry

Concept name	Organisational Structure
Definition	Organisational structure is a concept comprising various aspects and forms f structuring organisational units.
Description	Concepts used for describing various aspects and forms of structuring organisational units are categorised as belonging to the concept of organisational structure. Based on two different approaches, two criteria for classifying concepts of organisational structuring are used. The first depends on whether the given structure is the main structure or is it laid over the organisation, as a form of a superstructure. The second is based on the form of the structure, i.e. is it a hierarchical or heterarchical, or a mix of both.
Instance/s	Hierarchical, heterarchical

Table A.31: *Organisational Unit* data dictionary entry

Concept name	Organisational Unit (OU)
Definition	An organisational unit is the key elementary unit in the context of forming an organisation.
Description	An organisational unit is the elementary unit of an organisation that, under the influence of the other organisational concepts, forms an organisation. In the context of this document, and the area of LSMASs, an organisational unit is usually considered to represent an individual agent. Using the recursive definition though, an organisational unit that comprises multiple organisational units can be, under circumstances specified in [118], considered as an organisational unit. Using a more graphic explanation, a department organisational unit that comprises individual agents can be considered as individual organisational unit on a higher level of organisational hierarchy, where department organisational units form a higher-level organisational unit of a faculty.
Instance/s	maliAgent13
Attributes	definesRoles, hasRelation, hasRole, hasRelationship, definesRoles, hasCriteriaOfOrganizing, consistsOf, isPartOf

Table A.32: *Physical Artefact* data dictionary entry

Concept name	Physical Artefact
Synonyms	IVE Artefact
Definition	Physical artefacts are all the concepts that can be physically represented and included in a IVE.
Description	Every concept that describes objects that can be physically represented (e.g. a top hat), i.e. embodied and positioned on a topological map, and as such included in a IVE are classified as physical artefacts. Such elements have their role to play in the given IVE and usually contain a defined interface that governs the process of interaction of an agent with the given physical artefact.

Table A.33: *Physical Property* data dictionary entry

Concept name	Physical Property
Definition	
Description	Physical properties are key elements of physical artefacts, i.e. artefacts that can be visualised in a physical space. Usually when an artefact is used, a physical event is generated, and a physical property is modified.

Table A.34: *Plan* data dictionary entry

Concept name	Plan
Definition	A plan is a finite set of actions that leads to a specified goal.
Description	A plan is a finite set of actions that leads to a specified goal. An optimal plan cannot be made shorter if the same goal is retained in the process. The plan concept is especially useful when observing belief-desire-intention (BDI) agents, since it is driven by agents' desires and intentions.
Instance/s	How to solve the Quest for the DragonEgg

Table A.35: *Process* data dictionary entry

Concept name	Process (P)
Synonyms	Organisational Processes
Definition	A set of connected atomic actions.
Description	A process is in the context of this document defined as a set of atomic actions. Every process itself can be a part of another process, thus creating the recursive relation. A process can be performed in order for a goal to be met. It represents an activity or a procedure of an organisation [122].
Instance/s	RandomWalk

Table A.36: *Quest* data dictionary entry

Concept name	Quest (Q)
Definition	A quest is similar to a goal, but has a defined starting and ending situations.
Description	A quest is a similar to a goal, but it has a defined beginning and a defined end, i.e. a starting situation, and an ending situation[5]. In the context of MMORPGs, a quest is what drives a story, and, in principle, motivates the player to continue playing the game. Furthermore, a quest is often given to the player by an in-game character. A quest usually has various stages, and represents a challenge for the given player, thus embarking them on an adventure.
Instance/s	The Quest for the Dragon Egg

Table A.37: *Role* data dictionary entry

Concept name	Role (R)
Definition	A role is a set of norms with a common denominator.
Description	In the context of this document, a role is defined as a set of normative rules that are applicable to a particular part of the given organisation. Such normative rules are parts of the organisation's normative system, and can be grouped by specific criteria, thus forming roles. Roles are played by agents. When an agent plays a role, the role's constraints are applied to them, therefore constraining their possible actions, their perceivable goals, and their possibilities in general.
Instance/s	Wizard, Warrior, Ranged, Rogue
Attributes	isRoleIn, isRoleOf

Table A.38: *Rule* data dictionary entry

Concept name	Rule
Definition	Rules are elementary forms of constraints in normative systems, as they pose a basic aspect of defining standards.
Description	A rule is an atomic building block of a normative system. Rules are usually built in a general if-then form, meaning that two statements are connected with a causal link, thus regulating what happens (then part: consequent) if something else happens beforehand (if part: antecedent). Other forms of rules are possible as well, but are not used as often. For the most part, rules pose constraints on the given subject. Rules are commonly used for devising appropriate logical conditions for introducing modalities. [78]

Table A.39: *Situated Organisational Unit* data dictionary entry

Concept name	Situated Organisational Unit
Definition	Every organisational unit that is tied to a location through a situated norm is considered a situated organisational unit.
Description	An organisational unit that is tied to a specific IVE, or a specific geographic or otherwise place, is a situated organisational unit. Furthermore, such an organisational unit has some situated norms that refer to it. The place that is essential to the situated relation of a situated organisational unit can be physical or digital, but can usually be represented visually, following the description of an inhabitant agent.

Table A.40: *Strategic Alliance* data dictionary entry

Concept name	Strategic Alliance
Definition	Strategic alliance is a form of a long-lasting partnership of organisations of various forms, formed around a shared strategy, or a strategic goal.
Description	An alliance that is aimed at forming long-lasting partnerships consisting of organisations of various forms is dubbed a strategic alliance. A strategic alliance is formed around a strategy as a long-term objective that is shared amongst the strategic alliance members. Norms and regulations governing the expected behaviour within the strategic alliance are expected to be accepted by all the members, old and new alike.

Table A.41: *Strategy* data dictionary entry

Concept name **Synonyms**	Strategy Organisational Strategy
Definition	*Strategy defines the long term objectives of an organization, action plans for their realization as well as tools on how to measure success.* [122, 126]
Description	A strategy is, in the context of planning and shared organisational values, a long-term objective that is specified mosotly as a vision. It may consist of a number of objectives, quests, and similar. Strategy is therefore tentative in the context of plans of achieving it, but is versatile in terms of temporal likeness to change. Since it represents a long-term planning concept, a strategy is the main driving force of strategic alliances as agent coalitions meant to provide long-term suport to its members.

Table A.42: *Super Structure* data dictionary entry

Concept name	Super Structure
Definition	An inter-organisational structure formed above the conventional organisational structure.
Description	When organisations form structures comprising other organisations, a super-structure is formed. In the context of this document, a super-structure is thus described as an organisation of organisations, esentially spanning further than the usual reaches of a given average organisation. Such an inter-organisational structure is formed above the conventional organisational structure.

Table A.43: *Task* data dictionary entry

Concept name	Task
Definition	A task is the building block of a quest.
Description	A task is the building block of a quest, i.e. its elementary part. A quest is built of atomic tasks that are easier to follow in execution phase, rather than the overview provided by the main definition of a quest. In MMORPGs a quest could demand an item to be retrieved, yet such a simple-sounding quest could consist of various tasks that have to be fulfilled in order for the main quest to be finished. The relation of quest and task concepts can be recursive[6].

Table A.44: *Time Dependent Norm* data dictionary entry

Concept name	Time Dependent Norm
Definition	A norm that is dependent on the temporal aspect of the world is a time dependent norm.
Description	A time dependent norm is essentially a norm, but with an added temporal constraint. Particularly, a time dependent norm is constrained to a specific period in time, be it for its designated activity period, period during which the given norm is applicable, or simply the timeframe or a deadline when a change of the norm, or caused by the norm, is to be expected.
Instance/s	Every 24 hours the Dragon Egg item is created again, rendering the old one useless.

Table A.45: *Workspace* data dictionary entry

Concept name	Workspace (W)
Definition	A workspace is the union of all the elements of a system, including agents, artefacts, etc.
Description	A workspace is the complete environment of a given system, including all the agents, artefacts, etc. What sets the concept of a workspace apart from the concept of an environment is the extent of the involved concepts, i.e. a workspace contains all the elements of an organisation and the whole system, while environment comprises only the elements that are external to the given organisation. It is worth noting that elements of the environment are an integral part of the whole system, since the life and activities of the given organisation are influenced by them.

A.2 Instance Properties

Table A.46: *isAchievedBy* instance property table

Property name	isAchievedBy		
Description	What is the activity that can be used to achieve this particular goal is governed by this property. It further allows for inference on the topic of actions useful towards achieving a specific goal when an organisational unit is faced with achieving the given goal. Furthermore, knowing which action is to be undertaken in order to achieve the given goal, an organisational unit can reason and deduce the role it has to play, for it to have the particular action on its disposal.		
Value Type	**Domain**	**Range**	**Cardinality**
object	Objective	Behaviour, Agent action, Activity	1..1

Table A.47: *triggers* instance property table

Property name	triggers		
Description	Any goal can be a part of a greater chain of goals that are grouped into a quest, or an objective. Therefore this property can be used to determine that a goal triggers another goal that has to be achieved.		
Value Type	**Domain**	**Range**	**Cardinality**
object	Objective	Process	0..*

Table A.48: *isAccessibleTo* instance property table

Property name	isAccessibleTo		
Description	A knowledge artefact can be defined to be accessible to certain other concepts of a system, most notably any organisational unit or a role. Further in the metamodel the distinction between an individual knowledge artefact and an organisational knowledge artefact is introduced, along with its constraints. Since not all knowledge is available to and accessible by all the entities of a system, this property introduces further constraints on the mentioned.		
Value Type	**Domain**	**Range**	**Cardinality**
object	Knowledge artefact	Organisational unit, Role	0..*

Table A.49: *definesRoles* instance property table

Property name	definesRoles		
Description	Organisation is by definition a set of organisational units that can be described using various organisational features, but one of the distinctive featues is that an organisation can define various roles. These roles are to be played by organisational units of the given organisation, in order to achieve shared organisational goals.		
Value Type	**Domain**	**Range**	**Cardinality**
object	Organisation	Role	1..*

Table A.50: *hasCriteriaOfOrganizing* instance property table

Property name	hasCriteriaOfOrganizing		
Description	An organisation has to be motivated into existance using a criteria of organising. Such a criteria is what drived the included organisational units towards forming an organisation. In MMORPG domain, the most commmon criteria are quests, yet excellence can have a great effect on the process of organising and the structure of an organisation and its organisational units.		
Value Type	**Domain**	**Range**	**Cardinality**
object	Organisational unit, Process, Strategy	Criteria of organising	1..1

Table A.51: *isPartOf* instance property table

Property name	isPartOf		
Description	As per the definition of an organisatioanl unit laid out in [118], an organisational unit can represent either an individual agent, or a group of organisational units. Ultimately, since an organisational unit can comprise several organisational units, it may be a group of groups of agents. Therefore, this property is important in understanding the nature of a given organisational unit. Furthermore, various organisational features are applicable to the members of the given organisation, thus it is valuable to know explicity what are the organisational units included in a given organisation. Obviously, an organisational unit, i.e. an individual agent, can be isolated and work alone, not being a party of an organisatioanl unit of a higher level.		
Value Type	**Domain**	**Range**	**Cardinality**
object	Organisational unit	Organisational unit	0..*

Table A.52: *hasRole* instance property table

Property name	hasRole		
Description	Every organisational unit can play a number of roles at any given point in time. This property designates roles that are defined within an organisational unit, that are playable by its organisational units.		
Value Type	**Domain**	**Range**	**Cardinality**
object	Organisational unit	Role	0..*

Table A.53: *playsRole* instance property table

Property name	playsRole		
Description	Based on the norms of the given organisation, an organisational unit can play one or more roles simultaneously. This property describes which roles are played by an organisational unit at the given moment.		
Value Type	**Domain**	**Range**	**Cardinality**
object	Organisational unit	Role	0..*

Appendix B

Theoretical Background

B.1 Graphs

In general context of mathematics, a graph is a mathematical construct comprising a set of nodes and a set of edges between the nodes.

Formally [156], a graph G is defined by a finite set $V(G) : V(G) \neq \varnothing$, called vertices, and a finite set $E(G)$ that includes unordered pairs of distinct elements of $V(G)$ called edges. $V(G)$ is therefore called the vertex set, and $E(G)$ is called the edge set of G. Two vertices v, w are joined by an edge $\{v, w\}$.

Two graphs G_1 and G_2 are said to be isomorphic, $G_1 \cong G_2$ if their respective vertex sets and edges sets are corresponding, insomuch that *the number of edges joining any two vertices of G_1 is equal to the number of edges joining the corresponding vertices of G_2.* [156]

Should those edges have a direction, i.e. have designated source and target nodes, the given graph is a directed graph. A directed graph is thus defined analogously to a graph, with the key difference being the ordered pairs of distinct edges:

> "A *directed graph* or *digraph* D consists of a finite nonempty set V of points together with a prescribed collection X of ordered pairs of distinct points. The elements of X are *directed lines* or *arcs*." — Harary [58]

When labels are added to edges, thus rendering edges uniquely identifiable by four characteristics (source, target, label, index), the graph is a labeled graph.

> "A graph G is *labeled* when the p points are distinguished from one another by names such as $v_1, v_2, \ldots, v_p$." — Harary [58]

A graph that is a directed graph and all its nodes represent types, and all edges represent relationship types, is a typed graph.

> "A type graph is a combination of

- A set of nodes which may include data types
- A set of edges
- A source function from edges to nodes, which gives the source node of an edge
- A target function from edges to nodes, which gives the target node of an edge
- An inheritance relationship between nodes (a reflexive partial ordering)

” — Kleppe [69]

Using graph theory, a model can be defined as a number of constraints applied to a type graph.

> “A model is a combination of a type graph and a set of constraints of various types.”
> — Kleppe [69]

Continuing with graph theory, an instance of a model is a labeled graph the type of whose every node is a node in the model, and every edge’s *source and target are typed over the source and target of the edge’s type in the type graph* [69].

> “An instance of a model M is a labeled graph that can be typed over the type graph of M and satisfies all the constraints in M’s constraint set.” — Kleppe [69]

The following is the mentioned set of constraint types (further described in [69]): multiplicities, bidirectinality, ordering, uniqueness, acyclic, unshared, redefinition, subset, union.

B.2 Graph Grammars

This Appendix contains theoretical background necessary for clear understanding of the description of organisational dynamics in Section 2.2.2. The following overview of graph grammars mostly follows the account on graph grammars by Engelfriet and Rozenberg [39] and Corradini et al. [28].

Graph grammars are mechanisms that allow for mathematical modelling of graph transformations, with the main component being a finite set of productions. A production is defined as a triple (M, D, E), where M and D are graphs, and E is an embedding mechanism. A production is applied to graph H called a *host* if graph M occurs in H. A production is applied by (1) removing the occurence of M from H, (2) replacing it by D (or its isomorphic copy), and (3) attaching D to the remainder of H (denoted as H^-) using the defined embedding mechanism E. [39]

Two distinguishable types of embedding are gluing and connecting. As the name suggests, gluing requires that some parts of D, i.e. nodes or edges, are found in H^-, i.e. they are identified with some parts of M. Naturally, the identified parts have to be isomorphic. On the other hand, connecting creates new edges that are used for connecting D to H^- – such edges make the nodes from D and H^- neighbouring nodes. Edges between nodes in M and H are therefore removed when M is removed.

Two approaches stem from these two types of embedding: the gluing approach and the connecting approach. Based on the mathematical techniques used by a particular approach, they are known as the algebraic approach and the algorithmic approach, respectively. The approach used in this thesis, for the purposes of modelling organisational dynamics (Section 2.2.2), is algebraic approach, which is further detailed below. More specifically, the used graph grammars are of the node replacement type.

Node replacement graph grammars are described as a specific case of graph grammars where the mother graph M is a single node of the host graph H, although the daoughter graph D is still a graph. In other words, one is talking about local transformations, althogh iteration of the process leads to global transformation of the graph [39].

> "A typical, very simple, example of a node-replacement mechanism is the Node Label Controlled mechanism, or NLC mechanism. In the NLC framework one rewrites undirected node-labeled graphs. The productions are node-replacing productions and the embedding connection instructions connect the daughter graph to the neighbourhood of the mother node – hence the rewriting process is completely local. In the NLC approach "everything" is based on node labels."
>
> — Engelfriet and Rozenberg [39]

> "An NLC graph grammar is a system $G = (\sigma, \Delta, P, C, S)$ where $\Sigma - \Delta$ and Δ (with $\Delta \subseteq \Sigma$) are the alphabets of nonterminal and terminal node labels, respectively, P is a finite set of NLC productions, C is a connection relation, i.e., a binary relation over Σ, and S is the initial graph (usually with a single node)."
>
> — Engelfriet and Rozenberg [39]

The above excerpts from [39] state that NLC mechanism and NLC productions are to be used with undirected graphs. Graphs that are produced using the Lamrast$-+$ metamodel are directed. Therefore, an upgraded mechanism is needed, where direction of considered edges can be taken into account and expressed accordingly. Furthermore, NLC distinguishes types of nodes, based on their labels only. An upgrade is useful, where individual nodes can be distinguished – graph grammars with neighbourhood controlled embedding (NCE) [39].

The extension of NLC to directed graphs with labelled nodes is introduced simply by extending the NLC connection relation with edge direction.

> "The connection relation C now consists of triples (μ, δ, d) , where $d \in in, out$, to deal with the incoming edges and the outgoing edges of the mother node, respectively.

> These connection instructions are used in an obvious way. Thus, a connection instruction (μ, δ, in) means that the embedding process should establish an edge to each node labeled δ in the daughter graph D from each node labeled μ that is an "in-neighbour" of the mother node m (where the in-neighbours of m are all nodes n for which there is an edge from n to m in the host graph)."
>
> — Engelfriet and Rozenberg [39]

Further extension of the NLC mechanism is given as a dynamic edge relabeling.

> "This leads to connection instructions of the form $(\mu, p/q, \delta)$, where p and q are edge labels, and μ and δ are node labels as before. The meaning of this connection instruction is that the embedding process should establish an edge with label q between each μ-labeled p-neighbour of the mother node and each δ-labeled node in the daughter graph. Thus, edge label p is changed into edge label q."
>
> — Engelfriet and Rozenberg [39]

Finally, the extension that can work with both labelled edges (e) and a directed graph (d) in the context of neighbourhood controlled embedding, i.e. edNCE grammar [89, 88] referenced in [39], is defined in terms of productions and connection instructions as follows.

> "Each production of an edNCE grammar is of the form $X \to (D, C)$, and each connection instruction in C is of the form $(\mu, p/q, x, d)$, where μ is a node label, p and q are edge labels, x is a node of D, and $d \in \{\text{in, out}\}$. If, say, $d = \text{in}$, then this instruction is interpreted as follows: the embedding process should establish an edge with label q to node x of D from each μ-labeled p-neighbour of m that is an in-neighbour of m."
>
> — Engelfriet and Rozenberg [39]

Node replacement graph grammar type can be discussed in terms of its counterpart in the graph-replacement domain. Such a graph grammar, using the connecting approach (as opposed to gluing), if also discussed in [39]:

> "For an arbitrary graph grammar that uses the connecting approach to embedding, the productions of the grammar are of the form (M, D, C) where M and D are graphs (the mother and the daughter graph, respectively) and C is a set of connection instructions. Such an instruction is applied to a graph H by removing from H an induced subgraph (isomorphic to) M, replacing it by (a copy of) D, and embedding D in the remainder H^- of H by the connection instructions from C."
>
> — Engelfriet and Rozenberg [39]

This is further propagated to connection instructions for edNCE grammars, in the domain of graph-replacement graph grammars:

> "[...] for edNCE grammars a connection instruction is of the form $(m, \mu, p/q, x, d)$ with obvious meaning: a q-labeled edge should be established between x and every μ-labeled node of H^- that is a p-neighbour of m (preserving direction d)."
>
> — Engelfriet and Rozenberg [39]

Using formal definitions, one can define the above edNCE concepts as follows:

> "Let Σ be an alphabet of node labels and Γ an alphabet of edge labels. A graph over Σ and Γ is a tuple $H = (V, E, \lambda)$, where V is the finite set of nodes, $E \subseteq \{(v, \gamma, w) \| v, w \in V, v \neq w, \gamma \in \Gamma\}$ is the set of edges, and $\lambda : V \to \Sigma$ is the node labeling function.
>
> [...]
>
> A graph is undirected if for every $(u, \gamma, w) \in E$, also $(w, \gamma, u) \in E$.
>
> [...]
>
> graph with (neighbourhood controlled) embedding over Σ and Γ is a pair (H, C) with $H \in GR_{\Sigma,\Gamma}$ and $C \subseteq \Sigma \times \Gamma \times \Gamma \times V_H \times \{\text{in,out}\}$. C is the connection relation of (H, C), and each element $(\delta, \beta, \gamma, x, d)$ of C (with $\delta \in \Sigma$, $\beta, \gamma \in \Gamma$, $x \in V_H$, and $d \in \{\text{in,out}\}$) is a connection instruction of (H, C). To improve readability, a connection instruction $(\delta, \beta, \gamma, x, d)$ will always be written as $(\delta, \beta/\gamma, x, d)$."
>
> — Engelfriet and Rozenberg [39]

From the stated above, and in the light of the models constructed using the Lamrast−+ metamodel are directed graphs, it can be concluded that, $\forall (u, \gamma, w) : (u, \gamma, w) \in E \Rightarrow (w, \gamma, u) \notin E$

Since the Lamrast−+ metamodel creates graphs for which gluing approach is more useful, the algebraic approach is the more interesting one to be observed in more detail.

The basic element is again a production, i.e. a graph transformation rule, defined as $p : L \rightsquigarrow R$, where both L and R are graphs, on left- and right-hand side respectively. When there is a match m that fixes an occurrence of L in a given graph G, then the direct derivation where p is applied to G leading to a derived graph H is denoted as $G \overset{p,m}{\Longrightarrow} H$. Simply put, replacing the occurrence of L in G by R leads to H. Therefore it can be said that a graph production $p : L \rightsquigarrow R$ prescribes which nodes and edges are to be preserved, which deleted, and which created, by defining a partial correspondence between elements of its left- and right-hand sides. A production p has its graph homomorphism in match $m : L \to G$ which maps nodes and edges of L to G preserving graphical structure and the labels along the way. The relationship of the mentioned graphs thus far, and connected concepts, is as follows.

> "A match $m : L \to G$ for a production p is a graph homomorphism, mapping nodes and edges of L to G, in such a way that the graphical structure and the labels are

$$\begin{tikzcd}
p : (L \arrow[d, "m"] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] \arrow[dr, phantom, "(2)"] & R) \arrow[d, "m^*"] \\
p^* : (G & D \arrow[l, "l^*"] \arrow[r, "r^*"'] & H)
\end{tikzcd}$$

Figure B.1: More detailed direct derivation as a DPO construction, according to [28]

> preserved. The match $m_1 : L_1 \to G_1$ of the direct derivation (1) maps each element of L_1 to the element of G_1 carrying the same number. Applying production p_1 to graph G_1 at match m_1 we have to delete every object from G_1 which matches an element of L_1 that has no corresponding element in R_1 [...]. Symmetrically, we add to G_1 each element of R_1 that has no corresponding element in L_1 [...]."
>
> — Corradini et al. [28]

Therefore, when there all the nodes of L and R are the same, the situation is clear. Intuitively, when there are nodes in R that are not in L, these nodes have to be added to H. Contrariwise, nodes that are in L, but are not in R have to be removed from H.

Fig. B.1 shows schematic representation of the direct derivation from G to H which is a result of production p being applied to a match m, denoted by $d = (G \overset{p,m}{\Longrightarrow} H)$.

Two slightly different approaches are available in the domain of algebraic approaches, where direct derivations (rule applications) are modelled using gluing constructions of graphs. These constructions are formally characterised as pushouts having graphs as objects and graph homomorphisms as arrows. These two approaches are double-pushout (DPO), and single-pushout (SPO) approach. DPO is notably more strict than SPO, since it does not allow rewriting in problematic solutions where instructions are unclear or incomplete. A production in DPO is defined using a pair of graph homomorphisms, as follows.

> "A production in the DPO approach is given by a pair $L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ of graph homomorphisms from a common *interface graph* K, and a direct derivation consists of two gluing diagrams of graphs and total graph morphisms, as (1) and (2) in the diagram [in Fig. B.1]. The context graph D is obtained from the given graph G by deleting all elements of G which have a pre-image in L, but none in K. Via diagram (1) this deletion is described as an inverse gluing operation, while the second gluing diagram (2) models the actual insertion into H of all elements of R that do not have a pre-image in K."
>
> — Corradini et al. [28]

When a production is set as above, and DPO approach is observed, *the match m must satisfy an application condition, called the* the gluing condition. The mentioned condition is a set of two parts: dangling condition and identification condition.

In the context of DPO and the defined production of this approach, the following is a description of graph grammar system.

> "A *graph grammar* G consists of a set of productions P and a *start graph* G_0. A sequence of direct derivations $\rho = (G_0 \overset{p_1}{\Rightarrow} G_1 \overset{p_2}{\Rightarrow} \ldots \overset{p_n}{\Rightarrow} G_n)$ constitutes a *derivation* of the grammar, also denoted by $G_0 \Rightarrow^* G_n$. The *language* $\mathcal{L}(G)$ generated by the grammar G is the set of all graphs G_n such that $G_0 \Rightarrow^* G_n$, is a derivation of the grammar."
>
> — Corradini et al. [28]

Appendix C

Full Listings

C.1 Logical Production System

```
maxTime(20).

fluents
        skills(_,_,_,_),
        hasSkill(_,_,_),
        availableQuest(_),
        hasQuest(_,_),
        isAvailable(_),
        solvedQuest(_,_),
        party(_,_),
        questAvailable(_,_).
events
        makeAvailable(_),
        assignQuest(_,_).
actions
        modifySkill(_,_,_,_),
        assignQuest(_,_),
        solveQuest(_,_),
        initiateParty(_).

initially skills(alice,0,0,0), isAvailable(alice).
initially skills(bob, 0,0,0).
initially questAvailable(killMaggots,alice).
initially questAvailable(killMaggots,bob).

% player(Name, Agility, Strength, Intelligence).
player(bob).
player(alice).
% quest(Name, Duration).
% reward/requirement(Quest, Agility, Strength, Intelligence).
quest(killMaggots, 4).
```

```
reward(killMaggots, 1,1,0).
requirement(killMaggots, 0,0,0).
quest(seekPotion, 2).
reward(seekPotion, 0,1,1).
requirement(seekPotion, 0,1,0).
quest(dragonEgg, 6).
reward(dragonEgg, 3,1,0).
requirement(dragonEgg, 2,2,1).

follows(killMaggots, seekPotion).
follows(seekPotion, dragonEgg).

% stop validity of previous skill level,
% and initiate the new, increased by the given value
modifySkill(P,_,_,_)
        terminates skills(P, _, _, _).
modifySkill(P,L1,L2,L3) initiates skills(P,L1new,L2new,L3new)
if      skills(P,L1old,L2old,L3old),
        L1new is L1old + L1,
        L2new is L2old + L2,
        L3new is L3old + L3.
modifySkill(P,_,_,_) initiates isAvailable(P).

if isAvailable(P), quest(Q,_)
then considerQuest(P, Q) from T1 to T2.

considerQuest(P,Q) from T1 to T2 if
        skills(P,L1,L2,L3),
        quest(Q,_),
        questAvailable(Q,P),
        not solvedQuest(P,Q),
        requirement(Q,R1,R2,R3),
        L1 >= R1, L2 >= R2, L3 >= R3,
        goOnQuest(P,Q) from T1 to T2.

considerQuest(P,Q) from T1 to T2 if
        skills(P,L1,L2,L3),
        quest(Q,_),
        questAvailable(Q,P),
        not solvedQuest(P,Q),
        requirement(Q,R1,R2,R3),
        (L1 < R1; L2 < R2; L3 < R3),
        initiateParty(P) from T1 to T2.

% quest solving process - assign and solve the quest after Tq moments
goOnQuest(P,Q) from T1 to T4 if
        assignQuest(P,Q) from T1 to T2,
```

```
        quest(Q,Tq),
        player(P),
        T3 is T1 + Tq,
        solveQuest(P,Q) from T3 to T4.

assignQuest(P, Q)       initiates hasQuest(P, Q).
assignQuest(P, _)       terminates isAvailable(P).

solveQuest(P,Q) terminates hasQuest(P,Q).
solveQuest(P,Q) terminates questAvailable(Q,P).
solveQuest(P,Q) initiates solvedQuest(P,Q).
solveQuest(P,Q), follows(Q,Q1) initiates questAvailable(Q1,P).
if solveQuest(P,Q) from T1 to T2, reward(Q,L1,L2,L3)
then
        modifySkill(P,L1,L2,L3) from T2 to T3.

makeAvailable(P)
        initiates isAvailable(P).
initiateParty(P)
        initiates party(P,[P]).

false assignQuest(P,_), not isAvailable(P).
false assignQuest(P,Q), solvedQuest(P,Q).
false initiateParty(P), hasQuest(P,_).

observe makeAvailable(bob) from 3 to 4.
```

C.2 ZODB Object Definition

```
import persistent
import os


class savedNode(persistent.Persistent):
    """This is a class containing all the data specifying a Node in a
        specific ASG"""

    def __init__(self, coreAttrs):
        """Initialise the savedNode object with values for all the
            default attributes."""
        self.graphClass_ = coreAttrs[0]
        self.isClass = coreAttrs[1]
        self.in_connections_ = coreAttrs[2]
        self.out_connections_ = coreAttrs[3]
        self.containerFrame = coreAttrs[4]
        self.keyword_ = coreAttrs[5]
        self.editGGLabel = coreAttrs[6]
        self.GGset2Any = coreAttrs[7]
        self.GGLabel = coreAttrs[8]
        # self.rootNode = coreAttrs[9]
        self.objectNumber = coreAttrs[10]
        self.ID = coreAttrs[11]

    def saveAttributes(self, order, attrValues):
        """Save custom attributes of the Node."""
        self.realOrder = order
        self.attrs = attrValues

        print self.attrs

    def updateAttributes(self, attrValues, connections):
        """Update custom attributes of the Node."""
        self.attrs = attrValues

        print connections

        modelInCs = connections[0]
        modelOutCs = connections[1]

        for nodeType in modelInCs.keys():
            newConn = [
                x for x in modelInCs[nodeType]
                if x not in self.in_connections_[nodeType]]
            if len(newConn):
```

```
                self.in_connections_[nodeType].append(newConn[0])
                # print '{} added to {}'.format(newConn, self.attrs[self
                    .realOrder.index('name')])

        for nodeType in modelOutCs.keys():
            newConn = [
                x for x in modelOutCs[nodeType]
                if x not in self.out_connections_[nodeType]]
            if len(newConn):
                self.out_connections_[nodeType].append(newConn[0])
                # print '{} added to {}'.format(newConn, self.attrs[self
                    .realOrder.index('name')])

        print self.attrs

    def getAttribute(self, attrName):
        if hasattr(self, attrName):
            return self.attrs[self.realOrder.index(attrName)]

    def generateCodeSPADE(self, KB=None):
        """Generate code for the Node."""

        print "Generating stuff...", self.isClass

        # templates for agents ang behaviours
        agent = [
"""
class {0}(spade.Agent.Agent):
    '''Bear skeleton for agent type {0}'''
""",
"""
    def _setup(self):
        print '{0}: running'
        self.addBehaviour(self.ChangeRole(), None)

"""]
        behaviour = """
    class {0}(spade.Behaviour.OneShotBehaviour):
        '''Behaviour {0} of {2} {1}'''
        def _process(self):
            print '{1}: behaving {0}'
"""

        if hasattr(self, 'isClass') and self.isClass in ['OrgUnit']:
            # beginning of generated code
            code = "import spade\nfrom RoleBehaviours import *\n"

```

```
            nodeName = "OU{}{}".format(
                self.ID,
                self.attrs[self.realOrder.index('name')])

            file = open("./Code/{}.py".format(nodeName), 'w')

            # nodeName = "{}{}".format(
            #     self.isClass,
            #     self.attrs[self.realOrder.index('name')])

            code = code + agent[0].format(nodeName)

            print self.attrs[self.realOrder.index('hasActions')]

            for behav in self.attrs[self.realOrder.index('hasActions')].
               split("\n")[:-1]:
                # code = code + "\n{}\n".format(behav.getValue())
                code = code + behaviour.format(
                    behav,
                    self.attrs[self.realOrder.index('name')],
                    self.isClass)

            code = code + agent[1].format(nodeName)


            if KB:
                code = code + """
        self.configureKB('SWI', None, 'swipl')"""
                for x in KB:
                    code = code + """
        self.addBelieve('{0[1]}({0[0]},{0[2]})')""".format(x)

            file.write(code)
            file.close()

            print nodeName

            return nodeName
```

C.3 OWL Functional Syntax Ontology Rendering

```
Prefix(:=<http://www.semanticweb.org/bogdan/ontologies/2018/5/untitled-
    ontology-125#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
Prefix(OOVASIS:=<http://ai.foi.hr/modelmmorpg/ooooaflsmas.owl#>)
Prefix(MAM5:=<http://users.dsic.upv.es/%7ecarrasco/JaCalIVE_Ontology#>)
Prefix(MAMbO5:=<http://www.semanticweb.org/bogdan/ontologies/2016/11/
    MAMbO5#>)


Ontology(<http://www.semanticweb.org/bogdan/ontologies/2018/5/untitled-
    ontology-125>

Declaration(Class(<OOVASIS#AcademicStructure>))
Declaration(Class(<OOVASIS#AcquisitionStructure>))
Declaration(Class(<OOVASIS#Activity>))
Declaration(Class(<OOVASIS#AdhocracyStructure>))
Declaration(Class(<OOVASIS#Agent>))
Declaration(Class(<OOVASIS#AmoebaStructure>))
Declaration(Class(<OOVASIS#Behavior>))
Declaration(Class(<OOVASIS#BioteamingOrganization>))
Declaration(Class(<OOVASIS#BusinessProcessReengineering>))
Declaration(Class(<OOVASIS#ClientServerBehavior>))
Declaration(Class(<OOVASIS#ClusterStructure>))
Declaration(Class(<OOVASIS#CommunitiesOfPractice>))
Declaration(Class(<OOVASIS#ComplexAnalyticalMethod>))
Declaration(Class(<OOVASIS#CriteriaOfOrganizing>))
Declaration(Class(<OOVASIS#Culture>))
Declaration(Class(<OOVASIS#CultureRelation>))
Declaration(Class(<OOVASIS#CustomerOrientedStructure>))
Declaration(Class(<OOVASIS#DivisionalStructure>))
Declaration(Class(<OOVASIS#DynamicNetworkStructure>))
Declaration(Class(<OOVASIS#EmpoweredOrganization>))
Declaration(Class(<OOVASIS#FiniteStateMachineBehavior>))
Declaration(Class(<OOVASIS#FishnetStructure>))
Declaration(Class(<OOVASIS#FractalStructure>))
Declaration(Class(<OOVASIS#FrontBackStructure>))
Declaration(Class(<OOVASIS#FunctionalStructure>))
Declaration(Class(<OOVASIS#HeterarchicalStructure>))
Declaration(Class(<OOVASIS#HierarchicalStructure>))
Declaration(Class(<OOVASIS#HybridStructure>))
Declaration(Class(<OOVASIS#HypertextOrganization>))
```

```
Declaration(Class(<OOVASIS#InfiniteFlatHierarchyStructure>))
Declaration(Class(<OOVASIS#InternalMarketStructure>))
Declaration(Class(<OOVASIS#InvertedStructure>))
Declaration(Class(<OOVASIS#ItineraryBehavior>))
Declaration(Class(<OOVASIS#Kaizen>))
Declaration(Class(<OOVASIS#KnowledgeArtifact>))
Declaration(Class(<OOVASIS#LeanManagement>))
Declaration(Class(<OOVASIS#LearningOrganization>))
Declaration(Class(<OOVASIS#ListenerBehavior>))
Declaration(Class(<OOVASIS#MatrixStructure>))
Declaration(Class(<OOVASIS#MergerStructure>))
Declaration(Class(<OOVASIS#Norm>))
Declaration(Class(<OOVASIS#NormativeSystem>))
Declaration(Class(<OOVASIS#Objective>))
Declaration(Class(<OOVASIS#ObserverBehavior>))
Declaration(Class(<OOVASIS#OneShotBehavior>))
Declaration(Class(<OOVASIS#OpenOrganization>))
Declaration(Class(<OOVASIS#OrganizationalArchitecture>))
Declaration(Class(<OOVASIS#OrganizationalChange>))
Declaration(Class(<OOVASIS#OrganizationalCulture>))
Declaration(Class(<OOVASIS#OrganizationalDesignMethod>))
Declaration(Class(<OOVASIS#OrganizationalEnvironment>))
Declaration(Class(<OOVASIS#OrganizationalIndividuals>))
Declaration(Class(<OOVASIS#OrganizationalKnowledgeNetwork>))
Declaration(Class(<OOVASIS#OrganizationalMemory>))
Declaration(Class(<OOVASIS#OrganizationalProcesses>))
Declaration(Class(<OOVASIS#OrganizationalStrategy>))
Declaration(Class(<OOVASIS#OrganizationalStructure>))
Declaration(Class(<OOVASIS#OrganizationalUnit>))
Declaration(Class(<OOVASIS#ParallelBehavior>))
Declaration(Class(<OOVASIS#PeriodicBehavior>))
Declaration(Class(<OOVASIS#PlatformOrganization>))
Declaration(Class(<OOVASIS#Process>))
Declaration(Class(<OOVASIS#ProcessRelation>))
Declaration(Class(<OOVASIS#ProductDivisionalStructure>))
Declaration(Class(<OOVASIS#ProjectOrientedStructure>))
Declaration(Class(<OOVASIS#RelationValuePartition>))
Declaration(Class(<OOVASIS#Role>))
Declaration(Class(<OOVASIS#RoleFactoryBehavior>))
Declaration(Class(<OOVASIS#SequentialBehavior>))
Declaration(Class(<OOVASIS#ShamrockOrganization>))
Declaration(Class(<OOVASIS#SixSigma>))
Declaration(Class(<OOVASIS#StableSuperStructure>))
Declaration(Class(<OOVASIS#StarburstStructure>))
Declaration(Class(<OOVASIS#StaticNetworkStructure>))
Declaration(Class(<OOVASIS#StrategicAllianceStructure>))
Declaration(Class(<OOVASIS#StrategicOrganization>))
```

```
Declaration(Class(<OOVASIS#Strategy>))
Declaration(Class(<OOVASIS#StrategyRelation>))
Declaration(Class(<OOVASIS#StructuralRelation>))
Declaration(Class(<OOVASIS#SuperStructure>))
Declaration(Class(<OOVASIS#TaguchiMethod>))
Declaration(Class(<OOVASIS#TeamBasedStructure>))
Declaration(Class(<OOVASIS#TensorStructure>))
Declaration(Class(<OOVASIS#TeritorialStructure>))
Declaration(Class(<OOVASIS#TotalQualityManagement>))
Declaration(Class(<OOVASIS#ValuePartition>))
Declaration(Class(<OOVASIS#VirtualStructure>))
Declaration(Class(<MAM5#Action>))
Declaration(Class(<MAM5#Action_Rule>))
Declaration(Class(<MAM5#Agent>))
Declaration(Class(<MAM5#Agent_Action>))
Declaration(Class(<MAM5#Artifact>))
Declaration(Class(<MAM5#Human_Immersed_Agent>))
Declaration(Class(<MAM5#IVE>))
Declaration(Class(<MAM5#IVE_Artifact>))
Declaration(Class(<MAM5#IVE_Law>))
Declaration(Class(<MAM5#IVE_Law_Condition>))
Declaration(Class(<MAM5#IVE_Law_Type>))
Declaration(Class(<MAM5#IVE_Workspace>))
Declaration(Class(<MAM5#Inhabitant_Agent>))
Declaration(Class(<MAM5#Observable_Event>))
Declaration(Class(<MAM5#Observable_Property>))
Declaration(Class(<MAM5#Operation>))
Declaration(Class(<MAM5#Physical_Artifact>))
Declaration(Class(<MAM5#Physical_Event>))
Declaration(Class(<MAM5#Physical_Property>))
Declaration(Class(<MAM5#Plan>))
Declaration(Class(<MAM5#Signal>))
Declaration(Class(<MAM5#SimpleType>))
Declaration(Class(<MAM5#Smart_Resource_Artifact>))
Declaration(Class(<MAM5#Vector3D>))
Declaration(Class(<MAM5#Workspace>))
Declaration(Class(<MAMbO5#SituatedOrganizationalUnit>))
Declaration(Class(<MAMbO5#TimeDependentNorm>))
Declaration(ObjectProperty(<OOVASIS#accepts>))
Declaration(ObjectProperty(<OOVASIS#achieves>))
Declaration(ObjectProperty(<OOVASIS#definesRoles>))
Declaration(ObjectProperty(<OOVASIS#hasAccessTo>))
Declaration(ObjectProperty(<OOVASIS#hasChange>))
Declaration(ObjectProperty(<OOVASIS#hasCriteriaOfOrganizing>))
Declaration(ObjectProperty(<OOVASIS#hasCulture>))
Declaration(ObjectProperty(<OOVASIS#hasEnvironment>))
Declaration(ObjectProperty(<OOVASIS#hasIndividuals>))
```

```
Declaration(ObjectProperty(<OOVASIS#hasProcesses>))
Declaration(ObjectProperty(<OOVASIS#hasRelation>))
Declaration(ObjectProperty(<OOVASIS#hasRole>))
Declaration(ObjectProperty(<OOVASIS#hasStrategy>))
Declaration(ObjectProperty(<OOVASIS#hasStructure>))
Declaration(ObjectProperty(<OOVASIS#isAcceptedBy>))
Declaration(ObjectProperty(<OOVASIS#isAccessibleTo>))
Declaration(ObjectProperty(<OOVASIS#isAchievedBy>))
Declaration(ObjectProperty(<OOVASIS#isCriteriaOfOrganizingFor>))
Declaration(ObjectProperty(<OOVASIS#isPerformedBy>))
Declaration(ObjectProperty(<OOVASIS#isRelationOf>))
Declaration(ObjectProperty(<OOVASIS#isRoleIn>))
Declaration(ObjectProperty(<OOVASIS#isRoleOf>))
Declaration(ObjectProperty(<OOVASIS#isTriggeredBy>))
Declaration(ObjectProperty(<OOVASIS#modelIndividualsFor>))
Declaration(ObjectProperty(<OOVASIS#modelProcessesFor>))
Declaration(ObjectProperty(<OOVASIS#modelsChangeFor>))
Declaration(ObjectProperty(<OOVASIS#modelsCultureFor>))
Declaration(ObjectProperty(<OOVASIS#modelsEnvironmentFor>))
Declaration(ObjectProperty(<OOVASIS#modelsStrategyFor>))
Declaration(ObjectProperty(<OOVASIS#modelsStructureFor>))
Declaration(ObjectProperty(<OOVASIS#performs>))
Declaration(ObjectProperty(<OOVASIS#triggers>))
Declaration(ObjectProperty(<OOVASIS#usesAgents>))
Declaration(ObjectProperty(<OOVASIS#usesChange>))
Declaration(ObjectProperty(<OOVASIS#usesCulture>))
Declaration(ObjectProperty(<OOVASIS#usesEnvironment>))
Declaration(ObjectProperty(<OOVASIS#usesProcesses>))
Declaration(ObjectProperty(<OOVASIS#usesStrategy>))
Declaration(ObjectProperty(<OOVASIS#usesStructure>))
Declaration(ObjectProperty(<MAM5#generates_Signal>))
Declaration(ObjectProperty(<MAM5#has_Acceleration>))
Declaration(ObjectProperty(<MAM5#has_Action>))
Declaration(ObjectProperty(<MAM5#has_Action_Rule>))
Declaration(ObjectProperty(<MAM5#has_Agent>))
Declaration(ObjectProperty(<MAM5#has_Agent_Action>))
Declaration(ObjectProperty(<MAM5#has_Arguments>))
Declaration(ObjectProperty(<MAM5#has_Artifact>))
Declaration(ObjectProperty(<MAM5#has_Attribute>))
Declaration(ObjectProperty(<MAM5#has_Body_Artifact>))
Declaration(ObjectProperty(<MAM5#has_Component>))
Declaration(ObjectProperty(<MAM5#has_Do_Action>))
Declaration(ObjectProperty(<MAM5#has_IVE_Artifact>))
Declaration(ObjectProperty(<MAM5#has_IVE_Law>))
Declaration(ObjectProperty(<MAM5#has_IVE_Law_Cond_Type>))
Declaration(ObjectProperty(<MAM5#has_IVE_Law_Type>))
Declaration(ObjectProperty(<MAM5#has_IVE_Workspace>))
```

```
Declaration(ObjectProperty(<MAM5#has_Inh_Attribute>))
Declaration(ObjectProperty(<MAM5#has_Inhabitant_Agent>))
Declaration(ObjectProperty(<MAM5#has_Joint>))
Declaration(ObjectProperty(<MAM5#has_Observable_Property>))
Declaration(ObjectProperty(<MAM5#has_Operation>))
Declaration(ObjectProperty(<MAM5#has_Physical_Event>))
Declaration(ObjectProperty(<MAM5#has_Physical_Property>))
Declaration(ObjectProperty(<MAM5#has_Plan>))
Declaration(ObjectProperty(<MAM5#has_Position>))
Declaration(ObjectProperty(<MAM5#has_PreCondition>))
Declaration(ObjectProperty(<MAM5#has_Velocity>))
Declaration(ObjectProperty(<MAM5#has_Workspace>))
Declaration(ObjectProperty(<MAM5#is_Action_of>))
Declaration(ObjectProperty(<MAM5#is_Agent_Action_of>))
Declaration(ObjectProperty(<MAM5#is_Agent_of>))
Declaration(ObjectProperty(<MAM5#is_Artifact_of>))
Declaration(ObjectProperty(<MAM5#is_Body_Artifact_of>))
Declaration(ObjectProperty(<MAM5#is_Component_of>))
Declaration(ObjectProperty(<MAM5#is_IVE_Artifact_of>))
Declaration(ObjectProperty(<MAM5#is_IVE_Law_of>))
Declaration(ObjectProperty(<MAM5#is_IVE_Workspace_of>))
Declaration(ObjectProperty(<MAM5#is_Inhabitant_Agent_of>))
Declaration(ObjectProperty(<MAM5#is_Observable_Property_of>))
Declaration(ObjectProperty(<MAM5#is_Operation_of>))
Declaration(ObjectProperty(<MAM5#is_Physical_Property_of>))
Declaration(ObjectProperty(<MAM5#is_Plan_of>))
Declaration(ObjectProperty(<MAM5#is_Signal_generated_by>))
Declaration(ObjectProperty(<MAM5#is_Workspace_of>))
Declaration(ObjectProperty(<MAMbO5#EnvironmentIsUsedBy>))
Declaration(ObjectProperty(<MAMbO5#consistsOf>))
Declaration(ObjectProperty(<MAMbO5#hasActiveNorms>))
Declaration(ObjectProperty(<MAMbO5#isActiveWithin>))
Declaration(ObjectProperty(<MAMbO5#isPartOf>))
Declaration(ObjectProperty(<http://www.semanticweb.org/bogdan/ontologies
    /2017/4/MAMbO5ExampleScenario#playsRole>))
Declaration(DataProperty(<MAM5#Action>))
Declaration(DataProperty(<MAM5#Agent_Code_File>))
Declaration(DataProperty(<MAM5#Angle>))
Declaration(DataProperty(<MAM5#Artifact_Code_File>))
Declaration(DataProperty(<MAM5#Condition>))
Declaration(DataProperty(<MAM5#File>))
Declaration(DataProperty(<MAM5#IVE_Law_Action>))
Declaration(DataProperty(<MAM5#IVE_Law_Condition>))
Declaration(DataProperty(<MAM5#IVE_Law_Sentence>))
Declaration(DataProperty(<MAM5#IVE_Law_Type>))
Declaration(DataProperty(<MAM5#Linkeable>))
Declaration(DataProperty(<MAM5#Manual>))
```

```
Declaration(DataProperty(<MAM5#Mass>))
Declaration(DataProperty(<MAM5#Name>))
Declaration(DataProperty(<MAM5#Operand_Type>))
Declaration(DataProperty(<MAM5#Physical_Property_Type>))
Declaration(DataProperty(<MAM5#Shape>))
Declaration(DataProperty(<MAM5#X>))
Declaration(DataProperty(<MAM5#Y>))
Declaration(DataProperty(<MAM5#Z>))
Declaration(DataProperty(<MAM5#has_SimpleValue>))
Declaration(DataProperty(<MAMbO5#hasID>))
Declaration(DataProperty(<MAMbO5#isRelevantAtTime>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#ABattery>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#AElectricity>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#ATelevision>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Alice>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Bob>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#COInteraction>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Charlie>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Child>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Clerk>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Consumer>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Customer>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Diana>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Edgar>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Felipe>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Gonzalez>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUAcme>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUBlue>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUFamily>))
```

```
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUGreen>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUGreen3>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUNeighbourhood>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OURed>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OURed6>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OURoommates>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUShop>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUSmartBatteryGreen3>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUSmartBatteryRed6>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUSmartPVPanelGreen3>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUSmartPVPanelRed6>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#OUSmartTVRed6>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Parent>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Producer>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Roommate>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#Storage>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#StructuralRelation>))
Declaration(NamedIndividual(<http://www.semanticweb.org/bogdan/
    ontologies/2017/4/MAMbO5ExampleScenario#TopRole>))

############################
#   Object Properties
############################

# Object Property: <OOVASIS#accepts> (<OOVASIS#accepts>)

InverseObjectProperties(<OOVASIS#accepts> <OOVASIS#isAcceptedBy>)
ObjectPropertyDomain(<OOVASIS#accepts> <OOVASIS#NormativeSystem>)
ObjectPropertyRange(<OOVASIS#accepts> <OOVASIS#Behavior>)

```

```
# Object Property: <OOVASIS#achieves> (<OOVASIS#achieves>)

InverseObjectProperties(<OOVASIS#achieves> <OOVASIS#isAchievedBy>)
ObjectPropertyDomain(<OOVASIS#achieves> <OOVASIS#Activity>)
ObjectPropertyRange(<OOVASIS#achieves> <OOVASIS#Objective>)

# Object Property: <OOVASIS#definesRoles> (<OOVASIS#definesRoles>)

InverseObjectProperties(<OOVASIS#definesRoles> <OOVASIS#isRoleIn>)
AsymmetricObjectProperty(<OOVASIS#definesRoles>)
IrreflexiveObjectProperty(<OOVASIS#definesRoles>)
ObjectPropertyDomain(<OOVASIS#definesRoles> <OOVASIS#OrganizationalUnit>
    )
ObjectPropertyRange(<OOVASIS#definesRoles> <OOVASIS#Role>)

# Object Property: <OOVASIS#hasAccessTo> (<OOVASIS#hasAccessTo>)

InverseObjectProperties(<OOVASIS#hasAccessTo> <OOVASIS#isAccessibleTo>)
ObjectPropertyDomain(<OOVASIS#hasAccessTo> <OOVASIS#Agent>)
ObjectPropertyRange(<OOVASIS#hasAccessTo> <OOVASIS#KnowledgeArtifact>)

# Object Property: <OOVASIS#hasChange> (<OOVASIS#hasChange>)

InverseObjectProperties(<OOVASIS#hasChange> <OOVASIS#modelsChangeFor>)

# Object Property: <OOVASIS#hasCriteriaOfOrganizing> (<OOVASIS#
    hasCriteriaOfOrganizing>)

InverseObjectProperties(<OOVASIS#hasCriteriaOfOrganizing> <OOVASIS#
    isCriteriaOfOrganizingFor>)
FunctionalObjectProperty(<OOVASIS#hasCriteriaOfOrganizing>)
ObjectPropertyDomain(<OOVASIS#hasCriteriaOfOrganizing> ObjectUnionOf(<
    OOVASIS#OrganizationalUnit> <OOVASIS#Process> <OOVASIS#Strategy>))
ObjectPropertyRange(<OOVASIS#hasCriteriaOfOrganizing> <OOVASIS#
    CriteriaOfOrganizing>)

# Object Property: <OOVASIS#hasCulture> (<OOVASIS#hasCulture>)

InverseObjectProperties(<OOVASIS#hasCulture> <OOVASIS#modelsCultureFor>)

# Object Property: <OOVASIS#hasEnvironment> (<OOVASIS#hasEnvironment>)

InverseObjectProperties(<OOVASIS#hasEnvironment> <OOVASIS#
    modelsEnvironmentFor>)

# Object Property: <OOVASIS#hasIndividuals> (<OOVASIS#hasIndividuals>)

```

```
InverseObjectProperties(<OOVASIS#hasIndividuals> <OOVASIS#
    modelIndividualsFor>)

# Object Property: <OOVASIS#hasProcesses> (<OOVASIS#hasProcesses>)

InverseObjectProperties(<OOVASIS#hasProcesses> <OOVASIS#
    modelProcessesFor>)

# Object Property: <OOVASIS#hasRelation> (<OOVASIS#hasRelation>)

InverseObjectProperties(<OOVASIS#hasRelation> <OOVASIS#isRelationOf>)
FunctionalObjectProperty(<OOVASIS#hasRelation>)
ObjectPropertyRange(<OOVASIS#hasRelation> <OOVASIS#
    RelationValuePartition>)

# Object Property: <OOVASIS#hasRole> (<OOVASIS#hasRole>)

AnnotationAssertion(rdfs:comment <OOVASIS#hasRole> "Defines which roles
    can be played by which agents, i.e. organizational units, depending
    on the organization they are a part of, i.e. at any given point in
    time.")
InverseObjectProperties(<OOVASIS#hasRole> <OOVASIS#isRoleOf>)
AsymmetricObjectProperty(<OOVASIS#hasRole>)
IrreflexiveObjectProperty(<OOVASIS#hasRole>)
ObjectPropertyDomain(<OOVASIS#hasRole> <OOVASIS#OrganizationalUnit>)
ObjectPropertyRange(<OOVASIS#hasRole> <OOVASIS#Role>)

# Object Property: <OOVASIS#hasStrategy> (<OOVASIS#hasStrategy>)

InverseObjectProperties(<OOVASIS#hasStrategy> <OOVASIS#modelsStrategyFor
    >)

# Object Property: <OOVASIS#hasStructure> (<OOVASIS#hasStructure>)

InverseObjectProperties(<OOVASIS#hasStructure> <OOVASIS#
    modelsStructureFor>)

# Object Property: <OOVASIS#isCriteriaOfOrganizingFor> (<OOVASIS#
    isCriteriaOfOrganizingFor>)

InverseFunctionalObjectProperty(<OOVASIS#isCriteriaOfOrganizingFor>)
ObjectPropertyDomain(<OOVASIS#isCriteriaOfOrganizingFor> <OOVASIS#
    CriteriaOfOrganizing>)
ObjectPropertyRange(<OOVASIS#isCriteriaOfOrganizingFor> ObjectUnionOf(<
    OOVASIS#OrganizationalUnit> <OOVASIS#Process> <OOVASIS#Strategy>))

# Object Property: <OOVASIS#isPerformedBy> (<OOVASIS#isPerformedBy>)
```

```

InverseObjectProperties(<OOVASIS#isPerformedBy> <OOVASIS#performs>)
FunctionalObjectProperty(<OOVASIS#isPerformedBy>)
AsymmetricObjectProperty(<OOVASIS#isPerformedBy>)
IrreflexiveObjectProperty(<OOVASIS#isPerformedBy>)
ObjectPropertyDomain(<OOVASIS#isPerformedBy> <OOVASIS#Activity>)
ObjectPropertyRange(<OOVASIS#isPerformedBy> <OOVASIS#Agent>)

# Object Property: <OOVASIS#isRelationOf> (<OOVASIS#isRelationOf>)

InverseFunctionalObjectProperty(<OOVASIS#isRelationOf>)
ObjectPropertyDomain(<OOVASIS#isRelationOf> <OOVASIS#
    RelationValuePartition>)

# Object Property: <OOVASIS#isRoleIn> (<OOVASIS#isRoleIn>)

AsymmetricObjectProperty(<OOVASIS#isRoleIn>)
IrreflexiveObjectProperty(<OOVASIS#isRoleIn>)
ObjectPropertyDomain(<OOVASIS#isRoleIn> <OOVASIS#Role>)
ObjectPropertyRange(<OOVASIS#isRoleIn> <OOVASIS#OrganizationalUnit>)

# Object Property: <OOVASIS#isRoleOf> (<OOVASIS#isRoleOf>)

AsymmetricObjectProperty(<OOVASIS#isRoleOf>)
IrreflexiveObjectProperty(<OOVASIS#isRoleOf>)
ObjectPropertyDomain(<OOVASIS#isRoleOf> <OOVASIS#Role>)
ObjectPropertyRange(<OOVASIS#isRoleOf> <OOVASIS#OrganizationalUnit>)

# Object Property: <OOVASIS#isTriggeredBy> (<OOVASIS#isTriggeredBy>)

InverseObjectProperties(<OOVASIS#isTriggeredBy> <OOVASIS#triggers>)
ObjectPropertyDomain(<OOVASIS#isTriggeredBy> <OOVASIS#Process>)
ObjectPropertyRange(<OOVASIS#isTriggeredBy> <OOVASIS#Strategy>)

# Object Property: <OOVASIS#modelIndividualsFor> (<OOVASIS#
    modelIndividualsFor>)

ObjectPropertyDomain(<OOVASIS#modelIndividualsFor> <OOVASIS#
    OrganizationalIndividuals>)
ObjectPropertyRange(<OOVASIS#modelIndividualsFor> <OOVASIS#
    OrganizationalArchitecture>)

# Object Property: <OOVASIS#modelProcessesFor> (<OOVASIS#
    modelProcessesFor>)

ObjectPropertyDomain(<OOVASIS#modelProcessesFor> <OOVASIS#
    OrganizationalProcesses>)
```

```
ObjectPropertyRange(<OOVASIS#modelProcessesFor> <OOVASIS#OrganizationalArchitecture>)

# Object Property: <OOVASIS#modelsChangeFor> (<OOVASIS#modelsChangeFor>)

ObjectPropertyDomain(<OOVASIS#modelsChangeFor> <OOVASIS#OrganizationalChange>)
ObjectPropertyRange(<OOVASIS#modelsChangeFor> <OOVASIS#OrganizationalArchitecture>)

# Object Property: <OOVASIS#modelsCultureFor> (<OOVASIS#modelsCultureFor>)

ObjectPropertyDomain(<OOVASIS#modelsCultureFor> <OOVASIS#OrganizationalCulture>)
ObjectPropertyRange(<OOVASIS#modelsCultureFor> <OOVASIS#OrganizationalArchitecture>)

# Object Property: <OOVASIS#modelsEnvironmentFor> (<OOVASIS#modelsEnvironmentFor>)

ObjectPropertyDomain(<OOVASIS#modelsEnvironmentFor> <OOVASIS#OrganizationalEnvironment>)
ObjectPropertyRange(<OOVASIS#modelsEnvironmentFor> <OOVASIS#OrganizationalArchitecture>)

# Object Property: <OOVASIS#modelsStrategyFor> (<OOVASIS#modelsStrategyFor>)

ObjectPropertyDomain(<OOVASIS#modelsStrategyFor> <OOVASIS#OrganizationalStrategy>)
ObjectPropertyRange(<OOVASIS#modelsStrategyFor> <OOVASIS#OrganizationalArchitecture>)

# Object Property: <OOVASIS#modelsStructureFor> (<OOVASIS#modelsStructureFor>)

ObjectPropertyDomain(<OOVASIS#modelsStructureFor> <OOVASIS#OrganizationalStructure>)
ObjectPropertyRange(<OOVASIS#modelsStructureFor> <OOVASIS#OrganizationalArchitecture>)

# Object Property: <OOVASIS#performs> (<OOVASIS#performs>)

InverseFunctionalObjectProperty(<OOVASIS#performs>)
AsymmetricObjectProperty(<OOVASIS#performs>)
IrreflexiveObjectProperty(<OOVASIS#performs>)
```

```
ObjectPropertyDomain(<OOVASIS#performs> <OOVASIS#Agent>)
ObjectPropertyRange(<OOVASIS#performs> <OOVASIS#Activity>)

# Object Property: <OOVASIS#usesAgents> (<OOVASIS#usesAgents>)

ObjectPropertyDomain(<OOVASIS#usesAgents> <OOVASIS#
    OrganizationalIndividuals>)
ObjectPropertyRange(<OOVASIS#usesAgents> <OOVASIS#Agent>)

# Object Property: <OOVASIS#usesCulture> (<OOVASIS#usesCulture>)

ObjectPropertyDomain(<OOVASIS#usesCulture> <OOVASIS#
    OrganizationalCulture>)
ObjectPropertyRange(<OOVASIS#usesCulture> <OOVASIS#Culture>)

# Object Property: <OOVASIS#usesEnvironment> (<OOVASIS#usesEnvironment>)

InverseObjectProperties(<OOVASIS#usesEnvironment> <MAMb05#
    EnvironmentIsUsedBy>)
ObjectPropertyDomain(<OOVASIS#usesEnvironment> <OOVASIS#
    OrganizationalEnvironment>)
ObjectPropertyRange(<OOVASIS#usesEnvironment> <OOVASIS#Agent>)

# Object Property: <OOVASIS#usesProcesses> (<OOVASIS#usesProcesses>)

ObjectPropertyDomain(<OOVASIS#usesProcesses> <OOVASIS#
    OrganizationalProcesses>)
ObjectPropertyRange(<OOVASIS#usesProcesses> <OOVASIS#Process>)

# Object Property: <OOVASIS#usesStrategy> (<OOVASIS#usesStrategy>)

ObjectPropertyDomain(<OOVASIS#usesStrategy> <OOVASIS#
    OrganizationalStrategy>)
ObjectPropertyRange(<OOVASIS#usesStrategy> <OOVASIS#Strategy>)

# Object Property: <OOVASIS#usesStructure> (<OOVASIS#usesStructure>)

ObjectPropertyDomain(<OOVASIS#usesStructure> <OOVASIS#
    OrganizationalStructure>)
ObjectPropertyRange(<OOVASIS#usesStructure> <OOVASIS#OrganizationalUnit>
    )

# Object Property: <MAM5#generates_Signal> (<MAM5#generates_Signal>)

InverseObjectProperties(<MAM5#generates_Signal> <MAM5#
    is_Signal_generated_by>)
ObjectPropertyDomain(<MAM5#generates_Signal> <MAM5#Artifact>)
```

```
ObjectPropertyRange(<MAM5#generates_Signal> <MAM5#Signal>)

# Object Property: <MAM5#has_Acceleration> (<MAM5#has_Acceleration>)

ObjectPropertyDomain(<MAM5#has_Acceleration> <MAM5#Physical_Property>)
ObjectPropertyRange(<MAM5#has_Acceleration> <MAM5#Vector3D>)

# Object Property: <MAM5#has_Action> (<MAM5#has_Action>)

InverseObjectProperties(<MAM5#has_Action> <MAM5#is_Action_of>)
ObjectPropertyDomain(<MAM5#has_Action> <MAM5#IVE_Artifact>)
ObjectPropertyRange(<MAM5#has_Action> <MAM5#Action>)

# Object Property: <MAM5#has_Action_Rule> (<MAM5#has_Action_Rule>)

ObjectPropertyDomain(<MAM5#has_Action_Rule> <MAM5#Action>)
ObjectPropertyRange(<MAM5#has_Action_Rule> <MAM5#Action_Rule>)

# Object Property: <MAM5#has_Agent> (<MAM5#has_Agent>)

InverseObjectProperties(<MAM5#has_Agent> <MAM5#is_Agent_of>)
ObjectPropertyDomain(<MAM5#has_Agent> <MAM5#Workspace>)
ObjectPropertyRange(<MAM5#has_Agent> <MAM5#Agent>)

# Object Property: <MAM5#has_Agent_Action> (<MAM5#has_Agent_Action>)

InverseObjectProperties(<MAM5#has_Agent_Action> <MAM5#is_Agent_Action_of
   >)
ObjectPropertyDomain(<MAM5#has_Agent_Action> <MAM5#Agent>)
ObjectPropertyRange(<MAM5#has_Agent_Action> <MAM5#Agent_Action>)

# Object Property: <MAM5#has_Arguments> (<MAM5#has_Arguments>)

ObjectPropertyDomain(<MAM5#has_Arguments> <MAM5#Action>)

# Object Property: <MAM5#has_Artifact> (<MAM5#has_Artifact>)

InverseObjectProperties(<MAM5#has_Artifact> <MAM5#is_Artifact_of>)
ObjectPropertyDomain(<MAM5#has_Artifact> <MAM5#Workspace>)
ObjectPropertyRange(<MAM5#has_Artifact> <MAM5#Artifact>)

# Object Property: <MAM5#has_Attribute> (<MAM5#has_Attribute>)

ObjectPropertyDomain(<MAM5#has_Attribute> <MAM5#Artifact>)

# Object Property: <MAM5#has_Body_Artifact> (<MAM5#has_Body_Artifact>)

```

```
InverseObjectProperties(<MAM5#has_Body_Artifact> <MAM5#
    is_Body_Artifact_of>)
ObjectPropertyDomain(<MAM5#has_Body_Artifact> <MAM5#Inhabitant_Agent>)
ObjectPropertyRange(<MAM5#has_Body_Artifact> <MAM5#IVE_Artifact>)

# Object Property: <MAM5#has_Component> (<MAM5#has_Component>)

InverseObjectProperties(<MAM5#has_Component> <MAM5#is_Component_of>)
ObjectPropertyDomain(<MAM5#has_Component> <MAM5#IVE_Artifact>)
ObjectPropertyRange(<MAM5#has_Component> <MAM5#IVE_Artifact>)

# Object Property: <MAM5#has_Do_Action> (<MAM5#has_Do_Action>)

ObjectPropertyDomain(<MAM5#has_Do_Action> <MAM5#Action_Rule>)

# Object Property: <MAM5#has_IVE_Artifact> (<MAM5#has_IVE_Artifact>)

InverseObjectProperties(<MAM5#has_IVE_Artifact> <MAM5#is_IVE_Artifact_of
    >)
ObjectPropertyDomain(<MAM5#has_IVE_Artifact> <MAM5#IVE_Workspace>)
ObjectPropertyRange(<MAM5#has_IVE_Artifact> <MAM5#IVE_Artifact>)

# Object Property: <MAM5#has_IVE_Law> (<MAM5#has_IVE_Law>)

InverseObjectProperties(<MAM5#has_IVE_Law> <MAM5#is_IVE_Law_of>)
ObjectPropertyDomain(<MAM5#has_IVE_Law> <MAM5#IVE_Workspace>)
ObjectPropertyRange(<MAM5#has_IVE_Law> <MAM5#IVE_Law>)

# Object Property: <MAM5#has_IVE_Law_Cond_Type> (<MAM5#
    has_IVE_Law_Cond_Type>)

ObjectPropertyDomain(<MAM5#has_IVE_Law_Cond_Type> <MAM5#IVE_Law>)
ObjectPropertyRange(<MAM5#has_IVE_Law_Cond_Type> <MAM5#IVE_Law_Condition
    >)

# Object Property: <MAM5#has_IVE_Law_Type> (<MAM5#has_IVE_Law_Type>)

ObjectPropertyDomain(<MAM5#has_IVE_Law_Type> <MAM5#IVE_Law_Type>)
ObjectPropertyRange(<MAM5#has_IVE_Law_Type> ObjectUnionOf(<MAM5#
    SimpleType> <MAM5#Vector3D>))

# Object Property: <MAM5#has_IVE_Workspace> (<MAM5#has_IVE_Workspace>)

InverseObjectProperties(<MAM5#has_IVE_Workspace> <MAM5#
    is_IVE_Workspace_of>)
ObjectPropertyDomain(<MAM5#has_IVE_Workspace> <MAM5#IVE>)
ObjectPropertyRange(<MAM5#has_IVE_Workspace> <MAM5#IVE_Workspace>)
```

```
# Object Property: <MAM5#has_Inh_Attribute> (<MAM5#has_Inh_Attribute>)

ObjectPropertyDomain(<MAM5#has_Inh_Attribute> <MAM5#Inhabitant_Agent>)

# Object Property: <MAM5#has_Inhabitant_Agent> (<MAM5#
    has_Inhabitant_Agent>)

InverseObjectProperties(<MAM5#has_Inhabitant_Agent> <MAM5#
    is_Inhabitant_Agent_of>)
ObjectPropertyDomain(<MAM5#has_Inhabitant_Agent> <MAM5#IVE_Workspace>)
ObjectPropertyRange(<MAM5#has_Inhabitant_Agent> <MAM5#Inhabitant_Agent>)

# Object Property: <MAM5#has_Joint> (<MAM5#has_Joint>)

ObjectPropertyDomain(<MAM5#has_Joint> <MAM5#Physical_Property>)
ObjectPropertyRange(<MAM5#has_Joint> <MAM5#Vector3D>)

# Object Property: <MAM5#has_Observable_Property> (<MAM5#
    has_Observable_Property>)

InverseObjectProperties(<MAM5#has_Observable_Property> <MAM5#
    is_Observable_Property_of>)
ObjectPropertyDomain(<MAM5#has_Observable_Property> <MAM5#Artifact>)
ObjectPropertyRange(<MAM5#has_Observable_Property> <MAM5#
    Observable_Property>)

# Object Property: <MAM5#has_Operation> (<MAM5#has_Operation>)

InverseObjectProperties(<MAM5#has_Operation> <MAM5#is_Operation_of>)
ObjectPropertyDomain(<MAM5#has_Operation> <MAM5#Artifact>)
ObjectPropertyRange(<MAM5#has_Operation> <MAM5#Operation>)

# Object Property: <MAM5#has_Physical_Event> (<MAM5#has_Physical_Event>)

ObjectPropertyDomain(<MAM5#has_Physical_Event> <MAM5#Action>)
ObjectPropertyRange(<MAM5#has_Physical_Event> <MAM5#Physical_Event>)

# Object Property: <MAM5#has_Physical_Property> (<MAM5#
    has_Physical_Property>)

InverseObjectProperties(<MAM5#has_Physical_Property> <MAM5#
    is_Physical_Property_of>)
ObjectPropertyDomain(<MAM5#has_Physical_Property> <MAM5#IVE_Artifact>)
ObjectPropertyRange(<MAM5#has_Physical_Property> <MAM5#Physical_Property
    >)

```

```
# Object Property: <MAM5#has_Plan> (<MAM5#has_Plan>)

InverseObjectProperties(<MAM5#has_Plan> <MAM5#is_Plan_of>)
ObjectPropertyDomain(<MAM5#has_Plan> <MAM5#Agent>)
ObjectPropertyRange(<MAM5#has_Plan> <MAM5#Plan>)

# Object Property: <MAM5#has_Position> (<MAM5#has_Position>)

ObjectPropertyDomain(<MAM5#has_Position> <MAM5#Physical_Property>)
ObjectPropertyRange(<MAM5#has_Position> <MAM5#Vector3D>)

# Object Property: <MAM5#has_PreCondition> (<MAM5#has_PreCondition>)

ObjectPropertyDomain(<MAM5#has_PreCondition> <MAM5#Action_Rule>)

# Object Property: <MAM5#has_Velocity> (<MAM5#has_Velocity>)

ObjectPropertyDomain(<MAM5#has_Velocity> <MAM5#Physical_Property>)
ObjectPropertyRange(<MAM5#has_Velocity> <MAM5#Vector3D>)

# Object Property: <MAM5#has_Workspace> (<MAM5#has_Workspace>)

InverseObjectProperties(<MAM5#has_Workspace> <MAM5#is_Workspace_of>)
ObjectPropertyDomain(<MAM5#has_Workspace> <MAM5#IVE>)
ObjectPropertyRange(<MAM5#has_Workspace> <MAM5#Workspace>)

# Object Property: <MAM5#is_Action_of> (<MAM5#is_Action_of>)

ObjectPropertyDomain(<MAM5#is_Action_of> <MAM5#Action>)
ObjectPropertyRange(<MAM5#is_Action_of> <MAM5#IVE_Artifact>)

# Object Property: <MAM5#is_Agent_Action_of> (<MAM5#is_Agent_Action_of>)

ObjectPropertyDomain(<MAM5#is_Agent_Action_of> <MAM5#Agent_Action>)
ObjectPropertyRange(<MAM5#is_Agent_Action_of> <MAM5#Agent>)

# Object Property: <MAM5#is_Agent_of> (<MAM5#is_Agent_of>)

ObjectPropertyDomain(<MAM5#is_Agent_of> <MAM5#Agent>)
ObjectPropertyRange(<MAM5#is_Agent_of> <MAM5#Workspace>)

# Object Property: <MAM5#is_Artifact_of> (<MAM5#is_Artifact_of>)

ObjectPropertyDomain(<MAM5#is_Artifact_of> <MAM5#Artifact>)
ObjectPropertyRange(<MAM5#is_Artifact_of> <MAM5#Workspace>)
```

```
# Object Property: <MAM5#is_Body_Artifact_of> (<MAM5#is_Body_Artifact_of>)

ObjectPropertyDomain(<MAM5#is_Body_Artifact_of> <MAM5#IVE_Artifact>)
ObjectPropertyRange(<MAM5#is_Body_Artifact_of> <MAM5#Inhabitant_Agent>)

# Object Property: <MAM5#is_Component_of> (<MAM5#is_Component_of>)

ObjectPropertyDomain(<MAM5#is_Component_of> <MAM5#IVE_Artifact>)
ObjectPropertyRange(<MAM5#is_Component_of> <MAM5#IVE_Artifact>)

# Object Property: <MAM5#is_IVE_Artifact_of> (<MAM5#is_IVE_Artifact_of>)

ObjectPropertyDomain(<MAM5#is_IVE_Artifact_of> <MAM5#IVE_Artifact>)
ObjectPropertyRange(<MAM5#is_IVE_Artifact_of> <MAM5#IVE_Workspace>)

# Object Property: <MAM5#is_IVE_Law_of> (<MAM5#is_IVE_Law_of>)

ObjectPropertyDomain(<MAM5#is_IVE_Law_of> <MAM5#IVE_Law>)
ObjectPropertyRange(<MAM5#is_IVE_Law_of> <MAM5#IVE_Workspace>)

# Object Property: <MAM5#is_IVE_Workspace_of> (<MAM5#is_IVE_Workspace_of>)

ObjectPropertyDomain(<MAM5#is_IVE_Workspace_of> <MAM5#IVE_Workspace>)
ObjectPropertyRange(<MAM5#is_IVE_Workspace_of> <MAM5#IVE>)

# Object Property: <MAM5#is_Inhabitant_Agent_of> (<MAM5#is_Inhabitant_Agent_of>)

ObjectPropertyDomain(<MAM5#is_Inhabitant_Agent_of> <MAM5#Inhabitant_Agent>)
ObjectPropertyRange(<MAM5#is_Inhabitant_Agent_of> <MAM5#IVE_Workspace>)

# Object Property: <MAM5#is_Observable_Property_of> (<MAM5#is_Observable_Property_of>)

ObjectPropertyDomain(<MAM5#is_Observable_Property_of> <MAM5#Observable_Property>)
ObjectPropertyRange(<MAM5#is_Observable_Property_of> <MAM5#Artifact>)

# Object Property: <MAM5#is_Operation_of> (<MAM5#is_Operation_of>)

ObjectPropertyDomain(<MAM5#is_Operation_of> <MAM5#Operation>)
ObjectPropertyRange(<MAM5#is_Operation_of> <MAM5#Artifact>)

```

```
# Object Property: <MAM5#is_Physical_Property_of> (<MAM5#is_Physical_Property_of>)

ObjectPropertyDomain(<MAM5#is_Physical_Property_of> <MAM5#Physical_Property>)
ObjectPropertyRange(<MAM5#is_Physical_Property_of> <MAM5#IVE_Artifact>)

# Object Property: <MAM5#is_Plan_of> (<MAM5#is_Plan_of>)

ObjectPropertyDomain(<MAM5#is_Plan_of> <MAM5#Plan>)
ObjectPropertyRange(<MAM5#is_Plan_of> <MAM5#Agent>)

# Object Property: <MAM5#is_Signal_generated_by> (<MAM5#is_Signal_generated_by>)

ObjectPropertyDomain(<MAM5#is_Signal_generated_by> <MAM5#Signal>)
ObjectPropertyRange(<MAM5#is_Signal_generated_by> <MAM5#Artifact>)

# Object Property: <MAM5#is_Workspace_of> (<MAM5#is_Workspace_of>)

ObjectPropertyDomain(<MAM5#is_Workspace_of> <MAM5#Workspace>)
ObjectPropertyRange(<MAM5#is_Workspace_of> <MAM5#IVE>)

# Object Property: <MAMbO5#consistsOf> (<MAMbO5#consistsOf>)

InverseObjectProperties(<MAMbO5#consistsOf> <MAMbO5#isPartOf>)
ObjectPropertyDomain(<MAMbO5#consistsOf> <OOVASIS#OrganizationalUnit>)
ObjectPropertyRange(<MAMbO5#consistsOf> <OOVASIS#OrganizationalUnit>)

# Object Property: <MAMbO5#hasActiveNorms> (<MAMbO5#hasActiveNorms>)

InverseObjectProperties(<MAMbO5#hasActiveNorms> <MAMbO5#isActiveWithin>)

# Object Property: <MAMbO5#isActiveWithin> (<MAMbO5#isActiveWithin>)

ObjectPropertyDomain(<MAMbO5#isActiveWithin> <MAM5#IVE_Law>)
ObjectPropertyRange(<MAMbO5#isActiveWithin> <MAM5#IVE_Workspace>)

# Object Property: <http://www.semanticweb.org/bogdan/ontologies/2017/4/MAMbO5ExampleScenario#playsRole> (<http://www.semanticweb.org/bogdan/ontologies/2017/4/MAMbO5ExampleScenario#playsRole>)

AnnotationAssertion(rdfs:comment <http://www.semanticweb.org/bogdan/ontologies/2017/4/MAMbO5ExampleScenario#playsRole> "Defines which role is played by which agent at the moment of modelling.")
SubObjectPropertyOf(<http://www.semanticweb.org/bogdan/ontologies/2017/4/MAMbO5ExampleScenario#playsRole> <OOVASIS#hasRole>)
```

```
AsymmetricObjectProperty(<http://www.semanticweb.org/bogdan/ontologies
    /2017/4/MAMbO5ExampleScenario#playsRole>)
IrreflexiveObjectProperty(<http://www.semanticweb.org/bogdan/ontologies
    /2017/4/MAMbO5ExampleScenario#playsRole>)


############################
#   Data Properties
############################

# Data Property: <MAM5#Action> (<MAM5#Action>)

AnnotationAssertion(rdfs:comment <MAM5#Action> "Action as an effect-
    inducing function of an artefact."@en)
DataPropertyDomain(<MAM5#Action> <MAM5#IVE_Law>)
DataPropertyRange(<MAM5#Action> xsd:string)

# Data Property: <MAM5#Agent_Code_File> (<MAM5#Agent_Code_File>)

DataPropertyDomain(<MAM5#Agent_Code_File> <MAM5#Agent>)
DataPropertyRange(<MAM5#Agent_Code_File> xsd:string)

# Data Property: <MAM5#Angle> (<MAM5#Angle>)

DataPropertyDomain(<MAM5#Angle> <MAM5#Physical_Property>)
DataPropertyRange(<MAM5#Angle> xsd:float)

# Data Property: <MAM5#Artifact_Code_File> (<MAM5#Artifact_Code_File>)

DataPropertyDomain(<MAM5#Artifact_Code_File> <MAM5#Artifact>)
DataPropertyRange(<MAM5#Artifact_Code_File> xsd:string)

# Data Property: <MAM5#Condition> (<MAM5#Condition>)

DataPropertyDomain(<MAM5#Condition> <MAM5#IVE_Law>)
DataPropertyRange(<MAM5#Condition> xsd:string)

# Data Property: <MAM5#File> (<MAM5#File>)

DataPropertyDomain(<MAM5#File> owl:Thing)
DataPropertyRange(<MAM5#File> xsd:string)

# Data Property: <MAM5#IVE_Law_Action> (<MAM5#IVE_Law_Action>)

DataPropertyDomain(<MAM5#IVE_Law_Action> <MAM5#IVE_Law>)
DataPropertyRange(<MAM5#IVE_Law_Action> xsd:string)

```

```
# Data Property: <MAM5#IVE_Law_Condition> (<MAM5#IVE_Law_Condition>)

DataPropertyDomain(<MAM5#IVE_Law_Condition> <MAM5#IVE_Law_Condition>)
DataPropertyRange(<MAM5#IVE_Law_Condition> xsd:string)

# Data Property: <MAM5#IVE_Law_Sentence> (<MAM5#IVE_Law_Sentence>)

DataPropertyDomain(<MAM5#IVE_Law_Sentence> <MAM5#IVE_Law_Condition>)
DataPropertyRange(<MAM5#IVE_Law_Sentence> xsd:string)

# Data Property: <MAM5#IVE_Law_Type> (<MAM5#IVE_Law_Type>)

DataPropertyDomain(<MAM5#IVE_Law_Type> <MAM5#IVE_Law>)
DataPropertyRange(<MAM5#IVE_Law_Type> DataUnionOf(xsd:boolean xsd:double
    xsd:float xsd:int xsd:string))

# Data Property: <MAM5#Linkeable> (<MAM5#Linkeable>)

DataPropertyDomain(<MAM5#Linkeable> <MAM5#Artifact>)
DataPropertyRange(<MAM5#Linkeable> xsd:string)

# Data Property: <MAM5#Manual> (<MAM5#Manual>)

AnnotationAssertion(rdfs:comment <MAM5#Manual> "Used to define Artifacts
    and describe how to use them."@en)
DataPropertyDomain(<MAM5#Manual> <MAM5#Artifact>)
DataPropertyRange(<MAM5#Manual> xsd:string)

# Data Property: <MAM5#Mass> (<MAM5#Mass>)

DataPropertyDomain(<MAM5#Mass> <MAM5#Physical_Property>)
DataPropertyRange(<MAM5#Mass> xsd:float)

# Data Property: <MAM5#Name> (<MAM5#Name>)

DataPropertyDomain(<MAM5#Name> owl:Thing)
DataPropertyRange(<MAM5#Name> xsd:string)

# Data Property: <MAM5#Operand_Type> (<MAM5#Operand_Type>)

DataPropertyDomain(<MAM5#Operand_Type> owl:Thing)
DataPropertyRange(<MAM5#Operand_Type> DataOneOf("ADD" "AND" "BOOLEAN_VAL
   " "DIVIDE" "DOUBLE_VAL" "ELEMENT_ATT" "ELEMENT_PROP" "EQUAL" "
   FLOAT_VAL" "GREATERTHAN" "INT_VAL" "LESSTHAN" "MOD" "MULTIPLY" "OR" "
   PARAMETER" "STRING_VAL" "SUBSTRACT" "UNEQUAL"))

```

```
# Data Property: <MAM5#Physical_Property_Type> (<MAM5#
    Physical_Property_Type>)

DataPropertyDomain(<MAM5#Physical_Property_Type> <MAM5#Physical_Property
    >)
DataPropertyRange(<MAM5#Physical_Property_Type> DataOneOf("Internal" "
    Perceivable"))

# Data Property: <MAM5#Shape> (<MAM5#Shape>)

DataPropertyDomain(<MAM5#Shape> <MAM5#Physical_Property>)
DataPropertyRange(<MAM5#Shape> xsd:string)

# Data Property: <MAM5#X> (<MAM5#X>)

DataPropertyDomain(<MAM5#X> <MAM5#Vector3D>)
DataPropertyRange(<MAM5#X> xsd:float)

# Data Property: <MAM5#Y> (<MAM5#Y>)

DataPropertyDomain(<MAM5#Y> <MAM5#Vector3D>)
DataPropertyRange(<MAM5#Y> xsd:float)

# Data Property: <MAM5#Z> (<MAM5#Z>)

DataPropertyDomain(<MAM5#Z> <MAM5#Vector3D>)
DataPropertyRange(<MAM5#Z> xsd:float)

# Data Property: <MAM5#has_SimpleValue> (<MAM5#has_SimpleValue>)

DataPropertyDomain(<MAM5#has_SimpleValue> <MAM5#SimpleType>)
DataPropertyRange(<MAM5#has_SimpleValue> DataUnionOf(xsd:boolean
    xsd:double xsd:float xsd:integer xsd:string))

# Data Property: <MAMbO5#hasID> (<MAMbO5#hasID>)

SubDataPropertyOf(<MAMbO5#hasID> <MAM5#Name>)
FunctionalDataProperty(<MAMbO5#hasID>)

# Data Property: <MAMbO5#isRelevantAtTime> (<MAMbO5#isRelevantAtTime>)

DataPropertyDomain(<MAMbO5#isRelevantAtTime> <MAMbO5#TimeDependentNorm>)
DataPropertyRange(<MAMbO5#isRelevantAtTime> xsd:dateTime)



############################
```

```
#   Classes
############################

# Class: <OOVASIS#AcademicStructure> (<OOVASIS#AcademicStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#AcademicStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Akademska%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#AcademicStructure> <OOVASIS#HybridStructure>)

# Class: <OOVASIS#AcquisitionStructure> (<OOVASIS#AcquisitionStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#AcquisitionStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Spajanja%20i%20preuzimanja for details")
SubClassOf(<OOVASIS#AcquisitionStructure> <OOVASIS#SuperStructure>)

# Class: <OOVASIS#Activity> (<OOVASIS#Activity>)

AnnotationAssertion(rdfs:comment <OOVASIS#Activity> "Any atomic activity  performed by some individual agent
")
EquivalentClasses(<OOVASIS#Activity> <OOVASIS#Behavior>)
EquivalentClasses(<OOVASIS#Activity> <OOVASIS#Behavior> <MAM5#Agent_Action>)
SubClassOf(<OOVASIS#Activity> <OOVASIS#Process>)
SubClassOf(<OOVASIS#Activity> ObjectIntersectionOf(ObjectMinCardinality(1 <OOVASIS#achieves> <OOVASIS#Objective>) ObjectExactCardinality(1 <OOVASIS#isPerformedBy> <OOVASIS#Agent>)))
DisjointClasses(<OOVASIS#Activity> <OOVASIS#Agent>)
DisjointClasses(<OOVASIS#Activity> <OOVASIS#CriteriaOfOrganizing>)
DisjointClasses(<OOVASIS#Activity> <OOVASIS#OrganizationalUnit>)
DisjointClasses(<OOVASIS#Activity> <OOVASIS#Role>)

# Class: <OOVASIS#AdhocracyStructure> (<OOVASIS#AdhocracyStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#AdhocracyStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Ad-hoc%20suprastrukture%20(ad-hoc-kracije) for details")
SubClassOf(<OOVASIS#AdhocracyStructure> <OOVASIS#SuperStructure>)

# Class: <OOVASIS#Agent> (<OOVASIS#Agent>)

AnnotationAssertion(rdfs:comment <OOVASIS#Agent> "A person or thing (or piece of software of course) that takes an active role or produces a specified effect")
SubClassOf(<OOVASIS#Agent> <OOVASIS#OrganizationalUnit>)
```

```
SubClassOf(<OOVASIS#Agent> ObjectIntersectionOf(ObjectSomeValuesFrom(<
    OOVASIS#hasAccessTo> <OOVASIS#KnowledgeArtifact>)
    ObjectSomeValuesFrom(<OOVASIS#performs> <OOVASIS#Activity>)
    ObjectAllValuesFrom(<OOVASIS#hasAccessTo> <OOVASIS#KnowledgeArtifact>
    ) ObjectAllValuesFrom(<OOVASIS#performs> <OOVASIS#Activity>)))
DisjointClasses(<OOVASIS#Agent> <OOVASIS#CriteriaOfOrganizing>)
DisjointClasses(<OOVASIS#Agent> <OOVASIS#Process>)
DisjointClasses(<OOVASIS#Agent> <OOVASIS#Role>)

# Class: <OOVASIS#AmoebaStructure> (<OOVASIS#AmoebaStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#AmoebaStructure> "See http://
    ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Organizacijska%20struktura%20amebe for details")
SubClassOf(<OOVASIS#AmoebaStructure> <OOVASIS#AdhocracyStructure>)

# Class: <OOVASIS#Behavior> (<OOVASIS#Behavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#Behavior> "An agent behavior
    is some kind of activity performed by some agent. It has to be
    acceptable by a normative system the agent belongs to.")
EquivalentClasses(<OOVASIS#Behavior> <MAM5#Agent_Action>)
SubClassOf(<OOVASIS#Behavior> <OOVASIS#Process>)
SubClassOf(<OOVASIS#Behavior> ObjectIntersectionOf(ObjectSomeValuesFrom(
    <OOVASIS#isAcceptedBy> <OOVASIS#NormativeSystem>) ObjectAllValuesFrom
    (<OOVASIS#isAcceptedBy> <OOVASIS#NormativeSystem>)))

# Class: <OOVASIS#BioteamingOrganization> (<OOVASIS#
    BioteamingOrganization>)

AnnotationAssertion(rdfs:comment <OOVASIS#BioteamingOrganization> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Biotimovi for details")
SubClassOf(<OOVASIS#BioteamingOrganization> <OOVASIS#
    OrganizationalArchitecture>)

# Class: <OOVASIS#BusinessProcessReengineering> (<OOVASIS#
    BusinessProcessReengineering>)

AnnotationAssertion(rdfs:comment <OOVASIS#BusinessProcessReengineering>
    "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&
    page=Rein%C5%BEenjering%20poslovnih%20procesa for details")
SubClassOf(<OOVASIS#BusinessProcessReengineering> <OOVASIS#
    OrganizationalDesignMethod>)

# Class: <OOVASIS#ClientServerBehavior> (<OOVASIS#ClientServerBehavior>)

```

```
AnnotationAssertion(rdfs:comment <OOVASIS#ClientServerBehavior> "
    Behavior which resembles the client-server model, e.g. the client
    sends requests, the server responds to them")
SubClassOf(<OOVASIS#ClientServerBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#ClientServerBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#ClientServerBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#ClusterStructure> (<OOVASIS#ClusterStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#ClusterStructure> "See http://
    ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Klaster
    %20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#ClusterStructure> <OOVASIS#StableSuperStructure>)
DisjointClasses(<OOVASIS#ClusterStructure> <OOVASIS#StarburstStructure>)

# Class: <OOVASIS#CommunitiesOfPractice> (<OOVASIS#CommunitiesOfPractice
    >)

AnnotationAssertion(rdfs:comment <OOVASIS#CommunitiesOfPractice> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Dru%C5%A1tva%20razmjene%20najboljih%20praksi for details")
SubClassOf(<OOVASIS#CommunitiesOfPractice> <OOVASIS#
    OrganizationalDesignMethod>)

# Class: <OOVASIS#ComplexAnalyticalMethod> (<OOVASIS#
    ComplexAnalyticalMethod>)

AnnotationAssertion(rdfs:comment <OOVASIS#ComplexAnalyticalMethod> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Kompleksna%20analiti%C4%8Dka%20metoda for details")
SubClassOf(<OOVASIS#ComplexAnalyticalMethod> <OOVASIS#
    OrganizationalDesignMethod>)

# Class: <OOVASIS#CriteriaOfOrganizing> (<OOVASIS#CriteriaOfOrganizing>)

AnnotationAssertion(rdfs:comment <OOVASIS#CriteriaOfOrganizing> "A
    particular criteria for organizing things like processes,
    organizational units, strategies or cultural artifacts.")
DisjointClasses(<OOVASIS#CriteriaOfOrganizing> <OOVASIS#
    OrganizationalUnit>)
DisjointClasses(<OOVASIS#CriteriaOfOrganizing> <OOVASIS#Process>)
DisjointClasses(<OOVASIS#CriteriaOfOrganizing> <OOVASIS#Role>)

# Class: <OOVASIS#Culture> (<OOVASIS#Culture>)

AnnotationAssertion(rdfs:comment <OOVASIS#Culture> "Organizational
    culture in organizations is a complex cybernetic system that deals
```

```
with various intangible aspects of organizational behavior including but not limited to language, symbols, rituals, customs, norms, methods of problem solving, knowledge, learning etc.
")

# Class: <OOVASIS#CultureRelation> (<OOVASIS#CultureRelation>)

AnnotationAssertion(rdfs:comment <OOVASIS#CultureRelation> "A relation between cultural artifacts (e.g. knowledge, norms etc.) in the organizational culture perspective")
SubClassOf(<OOVASIS#CultureRelation> <OOVASIS#RelationValuePartition>)

# Class: <OOVASIS#CustomerOrientedStructure> (<OOVASIS#CustomerOrientedStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#CustomerOrientedStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Organizacijska%20struktura%20orijentirana%20prema%20potro%C5%A1a%C4%8Dima for details")
SubClassOf(<OOVASIS#CustomerOrientedStructure> <OOVASIS#DivisionalStructure>)

# Class: <OOVASIS#DivisionalStructure> (<OOVASIS#DivisionalStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#DivisionalStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Divizionalna%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#DivisionalStructure> <OOVASIS#HierarchicalStructure>)

# Class: <OOVASIS#DynamicNetworkStructure> (<OOVASIS#DynamicNetworkStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#DynamicNetworkStructure> "See:
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Dinami%C4%8Dna%20mre%C5%BEa
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=%C5%A0pageti%20organizacijska%20struktura
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Hollywoodska%20organizacijska%20struktura
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Umre%C5%BEena%20organizacijska%20struktura
for details")
SubClassOf(<OOVASIS#DynamicNetworkStructure> <OOVASIS#HeterarchicalStructure>)

```

```
# Class: <OOVASIS#EmpoweredOrganization> (<OOVASIS#EmpoweredOrganization
    >)

AnnotationAssertion(rdfs:comment <OOVASIS#EmpoweredOrganization> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Osna%C5%BEena%20organizacija for details")
SubClassOf(<OOVASIS#EmpoweredOrganization> <OOVASIS#
    OrganizationalArchitecture>)

# Class: <OOVASIS#FiniteStateMachineBehavior> (<OOVASIS#
    FiniteStateMachineBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#FiniteStateMachineBehavior> "A
     behavior which resembles a finite state machine in which every node
    is
an activity to be performed")
SubClassOf(<OOVASIS#FiniteStateMachineBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#FiniteStateMachineBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#FiniteStateMachineBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#FishnetStructure> (<OOVASIS#FishnetStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#FishnetStructure> "See http://
    ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Organizacijska%20struktura%20ribarske%20mre%C5%BEe for details")
SubClassOf(<OOVASIS#FishnetStructure> <OOVASIS#HeterarchicalStructure>)

# Class: <OOVASIS#FractalStructure> (<OOVASIS#FractalStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#FractalStructure> "See http://
    ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Fraktalna%20organizacijska%20struktura%20i%20koncept%20kaosa%20u%20
    organizaciji for details")
SubClassOf(<OOVASIS#FractalStructure> <OOVASIS#SuperStructure>)

# Class: <OOVASIS#FrontBackStructure> (<OOVASIS#FrontBackStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#FrontBackStructure> "See http:
    //ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Pramac/krma%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#FrontBackStructure> <OOVASIS#HybridStructure>)

# Class: <OOVASIS#FunctionalStructure> (<OOVASIS#FunctionalStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#FunctionalStructure> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Funkcionalna%20organizacijska%20struktura for details")
```

```
SubClassOf(<OOVASIS#FunctionalStructure> <OOVASIS#HierarchicalStructure>
    )

# Class: <OOVASIS#HeterarchicalStructure> (<OOVASIS#
    HeterarchicalStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#HeterarchicalStructure> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Heterarhijske%20strukture for details")
SubClassOf(<OOVASIS#HeterarchicalStructure> <OOVASIS#
    OrganizationalStructure>)

# Class: <OOVASIS#HierarchicalStructure> (<OOVASIS#HierarchicalStructure
    >)

AnnotationAssertion(rdfs:comment <OOVASIS#HierarchicalStructure> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Hijerarhijske%20strukture for details")
SubClassOf(<OOVASIS#HierarchicalStructure> <OOVASIS#
    OrganizationalStructure>)

# Class: <OOVASIS#HybridStructure> (<OOVASIS#HybridStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#HybridStructure> "See http://
    ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Hibridne%20strukture for details")
SubClassOf(<OOVASIS#HybridStructure> <OOVASIS#OrganizationalStructure>)

# Class: <OOVASIS#HypertextOrganization> (<OOVASIS#HypertextOrganization
    >)

AnnotationAssertion(rdfs:comment <OOVASIS#HypertextOrganization> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Hipertekst%20organizacija for details")
SubClassOf(<OOVASIS#HypertextOrganization> <OOVASIS#
    OrganizationalArchitecture>)

# Class: <OOVASIS#InfiniteFlatHierarchyStructure> (<OOVASIS#
    InfiniteFlatHierarchyStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#InfiniteFlatHierarchyStructure
    > "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=
    NULL&page=Beskona%C4%8Dno%20plitka%20organizacijska%20struktura for
    details")
SubClassOf(<OOVASIS#InfiniteFlatHierarchyStructure> <OOVASIS#
    HeterarchicalStructure>)
```

```
# Class: <OOVASIS#InternalMarketStructure> (<OOVASIS#
    InternalMarketStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#InternalMarketStructure> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Unutarnja%20tr%C5%BEi%C5%A1ta for details")
SubClassOf(<OOVASIS#InternalMarketStructure> <OOVASIS#
    HeterarchicalStructure>)

# Class: <OOVASIS#InvertedStructure> (<OOVASIS#InvertedStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#InvertedStructure> "See http:
    //ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Izvrnuta%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#InvertedStructure> <OOVASIS#HybridStructure>)

# Class: <OOVASIS#ItineraryBehavior> (<OOVASIS#ItineraryBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#ItineraryBehavior> "Behavior
    which allows mobile agents to travel across various locations and
    perform tasks")
SubClassOf(<OOVASIS#ItineraryBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#ItineraryBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#ItineraryBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#Kaizen> (<OOVASIS#Kaizen>)

AnnotationAssertion(rdfs:comment <OOVASIS#Kaizen> "See http://ai.foi.hr/
    oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Kaizen for
    details")
SubClassOf(<OOVASIS#Kaizen> <OOVASIS#OrganizationalDesignMethod>)

# Class: <OOVASIS#KnowledgeArtifact> (<OOVASIS#KnowledgeArtifact>)

AnnotationAssertion(rdfs:comment <OOVASIS#KnowledgeArtifact> "By
    knowledge artifact we understand a wide range of explicit knowledge
    in which we assume that it is queriable by the agent, including but
    not limited to data and knowledge bases, neural networks and machine
    learning architectures, various information services etc.
")
SubClassOf(<OOVASIS#KnowledgeArtifact> <OOVASIS#
    OrganizationalKnowledgeNetwork>)
SubClassOf(<OOVASIS#KnowledgeArtifact> <MAM5#Artifact>)
SubClassOf(<OOVASIS#KnowledgeArtifact> ObjectIntersectionOf(
    ObjectSomeValuesFrom(<OOVASIS#isAccessibleTo> <OOVASIS#Agent>)
    ObjectAllValuesFrom(<OOVASIS#isAccessibleTo> <OOVASIS#Agent>)))

```

```
# Class: <OOVASIS#LeanManagement> (<OOVASIS#LeanManagement>)

AnnotationAssertion(rdfs:comment <OOVASIS#LeanManagement> "See http://ai
    .foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Vitki%20
    menad%C5%BEment for details")
SubClassOf(<OOVASIS#LeanManagement> <OOVASIS#OrganizationalDesignMethod>
    )

# Class: <OOVASIS#LearningOrganization> (<OOVASIS#LearningOrganization>)

AnnotationAssertion(rdfs:comment <OOVASIS#LearningOrganization> "See
    http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Organizacija%20koja%20u%C4%8Di for details")
SubClassOf(<OOVASIS#LearningOrganization> <OOVASIS#
    OrganizationalArchitecture>)

# Class: <OOVASIS#ListenerBehavior> (<OOVASIS#ListenerBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#ListenerBehavior> "A special
    type of observer behavior in which and agent awaits a message of some
     other agent")
SubClassOf(<OOVASIS#ListenerBehavior> <OOVASIS#ObserverBehavior>)

# Class: <OOVASIS#MatrixStructure> (<OOVASIS#MatrixStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#MatrixStructure> "See http://
    ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Matri%
    C4%8Dna%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#MatrixStructure> <OOVASIS#HierarchicalStructure>)

# Class: <OOVASIS#MergerStructure> (<OOVASIS#MergerStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#MergerStructure> "See http://
    ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Spajanja%20i%20preuzimanja for details")
SubClassOf(<OOVASIS#MergerStructure> <OOVASIS#SuperStructure>)

# Class: <OOVASIS#Norm> (<OOVASIS#Norm>)

AnnotationAssertion(rdfs:comment <OOVASIS#Norm> "Norms are defined as (
    socially) accepted behavior in a defined group and represent a
    blueprint for behaving in said group")
SubClassOf(<OOVASIS#Norm> <OOVASIS#KnowledgeArtifact>)

# Class: <OOVASIS#NormativeSystem> (<OOVASIS#NormativeSystem>)

```

```
AnnotationAssertion(rdfs:comment <OOVASIS#NormativeSystem> "A normative system is a system of norms which apply to some organizational unit")
SubClassOf(<OOVASIS#NormativeSystem> <OOVASIS#OrganizationalKnowledgeNetwork>)

# Class: <OOVASIS#Objective> (<OOVASIS#Objective>)

AnnotationAssertion(rdfs:comment <OOVASIS#Objective> "Any measurable objective that can be achieved by an atomic activity. Objectives can trigger processes.
")
SubClassOf(<OOVASIS#Objective> <OOVASIS#Strategy>)
SubClassOf(<OOVASIS#Objective> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#isAchievedBy> <OOVASIS#Activity>) ObjectSomeValuesFrom(<OOVASIS#triggers> <OOVASIS#Process>) ObjectAllValuesFrom(<OOVASIS#triggers> <OOVASIS#Process>)))

# Class: <OOVASIS#ObserverBehavior> (<OOVASIS#ObserverBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#ObserverBehavior> "Behavior in which an agents awaits an event in order to perform its actions")
SubClassOf(<OOVASIS#ObserverBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#ObserverBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#ObserverBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#OneShotBehavior> (<OOVASIS#OneShotBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#OneShotBehavior> "A behavior which represents a simple task or activity which is stopped after performance")
SubClassOf(<OOVASIS#OneShotBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#OneShotBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#OneShotBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#OpenOrganization> (<OOVASIS#OpenOrganization>)

AnnotationAssertion(rdfs:comment <OOVASIS#OpenOrganization> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Otvorena%20organizacija for details")
SubClassOf(<OOVASIS#OpenOrganization> <OOVASIS#OrganizationalArchitecture>)

# Class: <OOVASIS#OrganizationalArchitecture> (<OOVASIS#OrganizationalArchitecture>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalArchitecture> "A model of an agent organization consisting of various perspectives
```

```
including structure, culture, processes, strategy and individuals.")
EquivalentClasses(<OOVASIS#OrganizationalArchitecture> ObjectIntersectionOf(ObjectMinCardinality(1 <OOVASIS#hasChange> <OOVASIS#OrganizationalChange>) ObjectMinCardinality(1 <OOVASIS#hasCulture> <OOVASIS#OrganizationalCulture>) ObjectMinCardinality(1 <OOVASIS#hasEnvironment> <OOVASIS#OrganizationalEnvironment>) ObjectMinCardinality(1 <OOVASIS#hasIndividuals> <OOVASIS#OrganizationalIndividuals>) ObjectMinCardinality(1 <OOVASIS#hasProcesses> <OOVASIS#OrganizationalProcesses>) ObjectMinCardinality(1 <OOVASIS#hasStrategy> <OOVASIS#OrganizationalStrategy>) ObjectMinCardinality(1 <OOVASIS#hasStructure> <OOVASIS#OrganizationalStructure>)))

# Class: <OOVASIS#OrganizationalChange> (<OOVASIS#OrganizationalChange>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalChange> "A model of organizational change in some agent organization (possibly influenced by some organizational design method)")
EquivalentClasses(<OOVASIS#OrganizationalChange> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#modelsChangeFor> <OOVASIS#OrganizationalArchitecture>) ObjectSomeValuesFrom(<OOVASIS#usesChange> <OOVASIS#OrganizationalDesignMethod>) ObjectAllValuesFrom(<OOVASIS#modelsChangeFor> <OOVASIS#OrganizationalArchitecture>)))

# Class: <OOVASIS#OrganizationalCulture> (<OOVASIS#OrganizationalCulture>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalCulture> "A model of an agent organization's culture")
EquivalentClasses(<OOVASIS#OrganizationalCulture> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#modelsCultureFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#modelsCultureFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#usesCulture> <OOVASIS#Culture>)))

# Class: <OOVASIS#OrganizationalDesignMethod> (<OOVASIS#OrganizationalDesignMethod>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalDesignMethod> "A method which brings change in and influences any part of an agent organization ")

# Class: <OOVASIS#OrganizationalEnvironment> (<OOVASIS#OrganizationalEnvironment>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalEnvironment> "A model of the organizational environment of some agent organization (
```

```
includes besides the environemnt the organization is located in also other organizations which are engaged in some way)")
AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalEnvironment> "Everything outside of the modelled system that can affect the modelled system. E.g. outside forces and agents that will not bemodelled in detail at the moment."@en)
EquivalentClasses(<OOVASIS#OrganizationalEnvironment> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#modelsEnvironmentFor> <OOVASIS#OrganizationalArchitecture>) ObjectSomeValuesFrom(<OOVASIS#usesEnvironment> <OOVASIS#Agent>) ObjectAllValuesFrom(<OOVASIS#modelsEnvironmentFor> <OOVASIS#OrganizationalArchitecture>)))

# Class: <OOVASIS#OrganizationalIndividuals> (<OOVASIS#OrganizationalIndividuals>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalIndividuals> "A model of an agent organization's individuals (agents)")
EquivalentClasses(<OOVASIS#OrganizationalIndividuals> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#modelIndividualsFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#modelIndividualsFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#usesAgents> <OOVASIS#Agent>)))

# Class: <OOVASIS#OrganizationalKnowledgeNetwork> (<OOVASIS#OrganizationalKnowledgeNetwork>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalKnowledgeNetwork> "Agent organizations can be seen as a network of knowledge artifacts which are accessible by particular agents. We will denote these with the label organizational knowldge network. Special cases of knowledge artifacts are norms which establish the rules of interaction between agents and values which influence decision making and selection of objectives
")
EquivalentClasses(<OOVASIS#OrganizationalKnowledgeNetwork> ObjectUnionOf(<OOVASIS#KnowledgeArtifact> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#hasRelation> <OOVASIS#CultureRelation>) ObjectAllValuesFrom(<OOVASIS#hasRelation> <OOVASIS#CultureRelation>) ObjectExactCardinality(1 <OOVASIS#hasCriteriaOfOrganizing> <OOVASIS#CriteriaOfOrganizing>))))
SubClassOf(<OOVASIS#OrganizationalKnowledgeNetwork> <OOVASIS#Culture>)

# Class: <OOVASIS#OrganizationalMemory> (<OOVASIS#OrganizationalMemory>)

```

```
AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalMemory> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Organizacijska%20memorija for details")
SubClassOf(<OOVASIS#OrganizationalMemory> <OOVASIS#OrganizationalDesignMethod>)

# Class: <OOVASIS#OrganizationalProcesses> (<OOVASIS#OrganizationalProcesses>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalProcesses> "A model of an agent organization's processes")
EquivalentClasses(<OOVASIS#OrganizationalProcesses> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#modelProcessesFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#modelProcessesFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#usesProcesses> <OOVASIS#Process>)))

# Class: <OOVASIS#OrganizationalStrategy> (<OOVASIS#OrganizationalStrategy>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalStrategy> "A model of an agent organization's strategy")
EquivalentClasses(<OOVASIS#OrganizationalStrategy> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#modelsStrategyFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#modelsStrategyFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#usesStrategy> <OOVASIS#Strategy>)))

# Class: <OOVASIS#OrganizationalStructure> (<OOVASIS#OrganizationalStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalStructure> "A model of an agent organization's structure")
EquivalentClasses(<OOVASIS#OrganizationalStructure> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#modelsStructureFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#modelsStructureFor> <OOVASIS#OrganizationalArchitecture>) ObjectAllValuesFrom(<OOVASIS#usesStructure> <OOVASIS#OrganizationalUnit>)))

# Class: <OOVASIS#OrganizationalUnit> (<OOVASIS#OrganizationalUnit>)

AnnotationAssertion(rdfs:comment <OOVASIS#OrganizationalUnit> "An organizational unit is (1) a network of agents (or lower level units), (2) which are organized according to some organizational criteria and (3) in which roles for lower level units are defined. This definition has an important implication: it allows us to deal with
```

```
agents, groups and teams of agents, organizations of agents, networks of organizations of agents (or organizations of organizations) as well as virtual organizations of agents (as overlay structures) in the same way. This in particular means that organizational units may form a lattice structure in which each unit can belong to several super-units and/or be composed of several subunits. The criteria of organizing could for example be an objective, function, goal, mission, unit name, higher-order role etc.
")
EquivalentClasses(<OOVASIS#OrganizationalUnit> ObjectUnionOf(<OOVASIS#Agent> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#definesRoles> <OOVASIS#Role>) ObjectSomeValuesFrom(<OOVASIS#hasRelation> <OOVASIS#StructuralRelation>) ObjectSomeValuesFrom(<OOVASIS#hasRole> <OOVASIS#Role>) ObjectAllValuesFrom(<OOVASIS#hasRelation> <OOVASIS#StructuralRelation>) ObjectMinCardinality(1 <OOVASIS#definesRoles> <OOVASIS#Role>) ObjectExactCardinality(1 <OOVASIS#hasCriteriaOfOrganizing> <OOVASIS#CriteriaOfOrganizing>))))
SubClassOf(<OOVASIS#OrganizationalUnit> <MAM5#Agent>)
DisjointClasses(<OOVASIS#OrganizationalUnit> <OOVASIS#Process>)
DisjointClasses(<OOVASIS#OrganizationalUnit> <OOVASIS#Role>)

# Class: <OOVASIS#ParallelBehavior> (<OOVASIS#ParallelBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#ParallelBehavior> "Various behaviors are run in parallel")
SubClassOf(<OOVASIS#ParallelBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#ParallelBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#ParallelBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#PeriodicBehavior> (<OOVASIS#PeriodicBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#PeriodicBehavior> "A behavior which is looped possibly with a given period of time intervals between iterations")
SubClassOf(<OOVASIS#PeriodicBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#PeriodicBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#PeriodicBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#PlatformOrganization> (<OOVASIS#PlatformOrganization>)

AnnotationAssertion(rdfs:comment <OOVASIS#PlatformOrganization> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Platformska%20organizacija for details")
SubClassOf(<OOVASIS#PlatformOrganization> <OOVASIS#OrganizationalArchitecture>)

# Class: <OOVASIS#Process> (<OOVASIS#Process>)
```

```

AnnotationAssertion(rdfs:comment <OOVASIS#Process> "A process is (1) a network of activities (or lower level processes) (2) according to some criteria of organizing and (3) triggered by some strategy. The given definition allows for modeling organizations as networks of processes which can be defined in a number of ways. For example, the criteria for organizing might be that one process uses inputs from another or that two processes are using the same resources, or even that two processes are performed by the same organizational unit or that they are crucial for the same organizational goal.
")
EquivalentClasses(<OOVASIS#Process> ObjectUnionOf(<OOVASIS#Activity> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#hasRelation> <OOVASIS#ProcessRelation>) ObjectSomeValuesFrom(<OOVASIS#isTriggeredBy> <OOVASIS#Strategy>) ObjectAllValuesFrom(<OOVASIS#hasRelation> <OOVASIS#ProcessRelation>) ObjectAllValuesFrom(<OOVASIS#isTriggeredBy> <OOVASIS#Strategy>) ObjectExactCardinality(1 <OOVASIS#hasCriteriaOfOrganizing> <OOVASIS#CriteriaOfOrganizing>))))
DisjointClasses(<OOVASIS#Process> <OOVASIS#Role>)

# Class: <OOVASIS#ProcessRelation> (<OOVASIS#ProcessRelation>)

AnnotationAssertion(rdfs:comment <OOVASIS#ProcessRelation> "A relation between two processes in the processes perspective")
SubClassOf(<OOVASIS#ProcessRelation> <OOVASIS#RelationValuePartition>)

# Class: <OOVASIS#ProductDivisionalStructure> (<OOVASIS#ProductDivisionalStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#ProductDivisionalStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Predmetna%20divizionalna%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#ProductDivisionalStructure> <OOVASIS#DivisionalStructure>)

# Class: <OOVASIS#ProjectOrientedStructure> (<OOVASIS#ProjectOrientedStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#ProjectOrientedStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Projektna%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#ProjectOrientedStructure> <OOVASIS#HierarchicalStructure>)

# Class: <OOVASIS#RelationValuePartition> (<OOVASIS#RelationValuePartition>)
```

```
AnnotationAssertion(rdfs:comment <OOVASIS#RelationValuePartition> "Value
     partition for the various organizational networks in some
    organizational architecture")
EquivalentClasses(<OOVASIS#RelationValuePartition> ObjectUnionOf(<
    OOVASIS#CultureRelation> <OOVASIS#ProcessRelation> <OOVASIS#
    StrategyRelation> <OOVASIS#StructuralRelation>))
SubClassOf(<OOVASIS#RelationValuePartition> <OOVASIS#ValuePartition>)

# Class: <OOVASIS#Role> (<OOVASIS#Role>)

AnnotationAssertion(rdfs:comment <OOVASIS#Role> "A prescribed or
    expected behavior associated with a particular position or status in
    a group or organization")
EquivalentClasses(<OOVASIS#Role> ObjectMinCardinality(1 <OOVASIS#
    isRoleIn> <OOVASIS#OrganizationalUnit>))
SubClassOf(<OOVASIS#Role> <OOVASIS#Norm>)

# Class: <OOVASIS#RoleFactoryBehavior> (<OOVASIS#RoleFactoryBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#RoleFactoryBehavior> "Behavior
     added at runtime and then enacted by the agent")
SubClassOf(<OOVASIS#RoleFactoryBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#RoleFactoryBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#RoleFactoryBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#SequentialBehavior> (<OOVASIS#SequentialBehavior>)

AnnotationAssertion(rdfs:comment <OOVASIS#SequentialBehavior> "A
    sequence of other behaviors")
SubClassOf(<OOVASIS#SequentialBehavior> <OOVASIS#Activity>)
SubClassOf(<OOVASIS#SequentialBehavior> <OOVASIS#Behavior>)
SubClassOf(<OOVASIS#SequentialBehavior> <MAM5#Agent_Action>)

# Class: <OOVASIS#ShamrockOrganization> (<OOVASIS#ShamrockOrganization>)

AnnotationAssertion(rdfs:comment <OOVASIS#ShamrockOrganization> "See:
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Organizacija%20djeteline
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Federalizam
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=
    Obrnuta%20krafna
for details")
SubClassOf(<OOVASIS#ShamrockOrganization> <OOVASIS#
    OrganizationalArchitecture>)
```

```
# Class: <OOVASIS#SixSigma> (<OOVASIS#SixSigma>)

AnnotationAssertion(rdfs:comment <OOVASIS#SixSigma> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=6%20%CF%83%20(Six%20Sigma) for details")
SubClassOf(<OOVASIS#SixSigma> <OOVASIS#OrganizationalDesignMethod>)

# Class: <OOVASIS#StableSuperStructure> (<OOVASIS#StableSuperStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#StableSuperStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Stabilne%20suprastrukture for details")
SubClassOf(<OOVASIS#StableSuperStructure> <OOVASIS#SuperStructure>)

# Class: <OOVASIS#StarburstStructure> (<OOVASIS#StarburstStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#StarburstStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Organizacijska%20struktura%20raspr%C5%A1ene%20zvijezde for details")
SubClassOf(<OOVASIS#StarburstStructure> <OOVASIS#StableSuperStructure>)

# Class: <OOVASIS#StaticNetworkStructure> (<OOVASIS#StaticNetworkStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#StaticNetworkStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Stati%C4%8Dna%20mre%C5%BEa for details")
SubClassOf(<OOVASIS#StaticNetworkStructure> <OOVASIS#HeterarchicalStructure>)

# Class: <OOVASIS#StrategicAllianceStructure> (<OOVASIS#StrategicAllianceStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#StrategicAllianceStructure> "See:
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Strate%C5%A1ki%20savezi%20i%20alijanse
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Internetski%20savezi
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Keiretsu
http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Chaebol
for details")
SubClassOf(<OOVASIS#StrategicAllianceStructure> <OOVASIS#SuperStructure>)

```

```
# Class: <OOVASIS#StrategicOrganization> (<OOVASIS#StrategicOrganization>)

AnnotationAssertion(rdfs:comment <OOVASIS#StrategicOrganization> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Strategijska%20organizacija for details")
SubClassOf(<OOVASIS#StrategicOrganization> <OOVASIS#OrganizationalArchitecture>)

# Class: <OOVASIS#Strategy> (<OOVASIS#Strategy>)

AnnotationAssertion(rdfs:comment <OOVASIS#Strategy> "Strategy is closely bound the the Balanced ScoreCard paradigm. A strategy consists of: (1) a network of objectives (or other smaller strategies), (2) a criteria of organizing this network e.g. criteria might be influence (the outcome of one strategy influences another, for example a mathematical function), responsibility (two strategies are under the responsibility of the same organizational unit), achieveability (two strategies can be achieved by the same organizational process), etc., (3) a process which is triggered from the strategy as a response to some environmental or internal change.
")
EquivalentClasses(<OOVASIS#Strategy> ObjectUnionOf(<OOVASIS#Objective> ObjectIntersectionOf(ObjectSomeValuesFrom(<OOVASIS#hasRelation> <OOVASIS#StrategyRelation>) ObjectSomeValuesFrom(<OOVASIS#triggers> <OOVASIS#Process>) ObjectAllValuesFrom(<OOVASIS#hasRelation> <OOVASIS#StrategyRelation>) ObjectAllValuesFrom(<OOVASIS#triggers> <OOVASIS#Process>) ObjectExactCardinality(1 <OOVASIS#hasCriteriaOfOrganizing> <OOVASIS#CriteriaOfOrganizing>))))

# Class: <OOVASIS#StrategyRelation> (<OOVASIS#StrategyRelation>)

AnnotationAssertion(rdfs:comment <OOVASIS#StrategyRelation> "A relation between two strategies in the strategic perspective")
SubClassOf(<OOVASIS#StrategyRelation> <OOVASIS#RelationValuePartition>)

# Class: <OOVASIS#StructuralRelation> (<OOVASIS#StructuralRelation>)

AnnotationAssertion(rdfs:comment <OOVASIS#StructuralRelation> "A relation between two organizational units in the organizational structure perspective")
SubClassOf(<OOVASIS#StructuralRelation> <OOVASIS#RelationValuePartition>)

# Class: <OOVASIS#SuperStructure> (<OOVASIS#SuperStructure>)

```

```
AnnotationAssertion(rdfs:comment <OOVASIS#SuperStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Suprastrukture for details")
SubClassOf(<OOVASIS#SuperStructure> <OOVASIS#OrganizationalStructure>)

# Class: <OOVASIS#TaguchiMethod> (<OOVASIS#TaguchiMethod>)

AnnotationAssertion(rdfs:comment <OOVASIS#TaguchiMethod> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Taguchi%20metoda for details")
SubClassOf(<OOVASIS#TaguchiMethod> <OOVASIS#OrganizationalDesignMethod>)

# Class: <OOVASIS#TeamBasedStructure> (<OOVASIS#TeamBasedStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#TeamBasedStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Timska%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#TeamBasedStructure> <OOVASIS#AdhocracyStructure>)

# Class: <OOVASIS#TensorStructure> (<OOVASIS#TensorStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#TensorStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Tenzorska%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#TensorStructure> <OOVASIS#HierarchicalStructure>)

# Class: <OOVASIS#TeritorialStructure> (<OOVASIS#TeritorialStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#TeritorialStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Teritorijalna%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#TeritorialStructure> <OOVASIS#DivisionalStructure>)

# Class: <OOVASIS#TotalQualityManagement> (<OOVASIS#TotalQualityManagement>)

AnnotationAssertion(rdfs:comment <OOVASIS#TotalQualityManagement> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Cjelovito%20upravljanje%20kvalitetom for details")
SubClassOf(<OOVASIS#TotalQualityManagement> <OOVASIS#OrganizationalDesignMethod>)

# Class: <OOVASIS#ValuePartition> (<OOVASIS#ValuePartition>)

AnnotationAssertion(rdfs:comment <OOVASIS#ValuePartition> "Value partitions")

```

```
# Class: <OOVASIS#VirtualStructure> (<OOVASIS#VirtualStructure>)

AnnotationAssertion(rdfs:comment <OOVASIS#VirtualStructure> "See http://ai.foi.hr/oovasis/wiki/wiki.php?name=OOVASIS&parent=NULL&page=Virtualna%20organizacijska%20struktura for details")
SubClassOf(<OOVASIS#VirtualStructure> <OOVASIS#AdhocracyStructure>)

# Class: <MAM5#Action> (<MAM5#Action>)

SubClassOf(<MAM5#Action> ObjectMinCardinality(1 <MAM5#has_Action_Rule> <MAM5#Action_Rule>))
SubClassOf(<MAM5#Action> ObjectMinCardinality(0 <MAM5#has_Physical_Event> <MAM5#Physical_Event>))

# Class: <MAM5#Action_Rule> (<MAM5#Action_Rule>)

SubClassOf(<MAM5#Action_Rule> ObjectMinCardinality(1 <MAM5#has_Do_Action>))
SubClassOf(<MAM5#Action_Rule> ObjectMinCardinality(0 <MAM5#has_PreCondition>))

# Class: <MAM5#Agent_Action> (<MAM5#Agent_Action>)

SubClassOf(<MAM5#Agent_Action> <OOVASIS#Process>)

# Class: <MAM5#Human_Immersed_Agent> (<MAM5#Human_Immersed_Agent>)

SubClassOf(<MAM5#Human_Immersed_Agent> <MAM5#Inhabitant_Agent>)

# Class: <MAM5#IVE> (<MAM5#IVE>)

AnnotationAssertion(rdfs:comment <MAM5#IVE> "Intelligent Virtual Environment Definition")

# Class: <MAM5#IVE_Artifact> (<MAM5#IVE_Artifact>)

SubClassOf(<MAM5#IVE_Artifact> <MAM5#Artifact>)

# Class: <MAM5#IVE_Law> (<MAM5#IVE_Law>)

AnnotationAssertion(rdfs:comment <MAM5#IVE_Law> "A type of norm that is dependent on a specific Workspace, i.e. it is location-based."@en)
EquivalentClasses(<MAM5#IVE_Law> ObjectIntersectionOf(<OOVASIS#Norm> ObjectSomeValuesFrom(<MAM5#is_IVE_Law_of> <MAM5#IVE_Workspace>) ObjectAllValuesFrom(<MAM5#is_IVE_Law_of> <MAM5#IVE_Workspace>)))
SubClassOf(<MAM5#IVE_Law> <OOVASIS#Norm>)
```

```
SubClassOf(<MAM5#IVE_Law> DataMinCardinality(1 <MAM5#IVE_Law_Action>
    xsd:string))

# Class: <MAM5#IVE_Law_Condition> (<MAM5#IVE_Law_Condition>)

EquivalentClasses(<MAM5#IVE_Law_Condition> <MAM5#IVE_Law_Type>)

# Class: <MAM5#IVE_Workspace> (<MAM5#IVE_Workspace>)

SubClassOf(<MAM5#IVE_Workspace> <MAM5#Workspace>)

# Class: <MAM5#Inhabitant_Agent> (<MAM5#Inhabitant_Agent>)

SubClassOf(<MAM5#Inhabitant_Agent> <MAM5#Agent>)
SubClassOf(<MAM5#Inhabitant_Agent> <MAMbO5#SituatedOrganizationalUnit>)

# Class: <MAM5#Physical_Artifact> (<MAM5#Physical_Artifact>)

SubClassOf(<MAM5#Physical_Artifact> <MAM5#IVE_Artifact>)

# Class: <MAM5#Physical_Event> (<MAM5#Physical_Event>)

SubClassOf(<MAM5#Physical_Event> <MAM5#Observable_Event>)

# Class: <MAM5#Physical_Property> (<MAM5#Physical_Property>)

SubClassOf(<MAM5#Physical_Property> <MAM5#Observable_Property>)

# Class: <MAM5#Plan> (<MAM5#Plan>)

SubClassOf(<MAM5#Plan> <OOVASIS#Strategy>)

# Class: <MAM5#SimpleType> (<MAM5#SimpleType>)

EquivalentClasses(<MAM5#SimpleType> <MAM5#Vector3D>)

# Class: <MAM5#Smart_Resource_Artifact> (<MAM5#Smart_Resource_Artifact>)

SubClassOf(<MAM5#Smart_Resource_Artifact> <MAM5#Physical_Artifact>)

# Class: <MAM5#Workspace> (<MAM5#Workspace>)

AnnotationAssertion(rdfs:comment <MAM5#Workspace> "Everything that is
    being modelled at the moment. May contain Organizational Units (
    Individual and Grouped). Does not contain concepts of the system that
     are not being modelled at the moment."@en)

```

```
# Class: <MAMbO5#SituatedOrganizationalUnit> (<MAMbO5#
    SituatedOrganizationalUnit>)

EquivalentClasses(<MAMbO5#SituatedOrganizationalUnit> ObjectUnionOf(<
    MAM5#Inhabitant_Agent> ObjectIntersectionOf(<OOVASIS#
    OrganizationalUnit> ObjectSomeValuesFrom(<MAM5#has_IVE_Law> <MAM5#
    IVE_Law>) ObjectAllValuesFrom(<MAM5#has_IVE_Law> <MAM5#IVE_Law>))))
SubClassOf(<MAMbO5#SituatedOrganizationalUnit> <OOVASIS#
    OrganizationalUnit>)

# Class: <MAMbO5#TimeDependentNorm> (<MAMbO5#TimeDependentNorm>)

EquivalentClasses(<MAMbO5#TimeDependentNorm> ObjectIntersectionOf(<
    OOVASIS#Norm> DataSomeValuesFrom(<MAMbO5#isRelevantAtTime>
    xsd:dateTime) DataAllValuesFrom(<MAMbO5#isRelevantAtTime>
    xsd:dateTime)))
SubClassOf(<MAMbO5#TimeDependentNorm> <OOVASIS#Norm>)



DisjointClasses(<OOVASIS#AcademicStructure> <OOVASIS#FrontBackStructure>
     <OOVASIS#InvertedStructure>)
DisjointClasses(<OOVASIS#AcquisitionStructure> <OOVASIS#
    AdhocracyStructure> <OOVASIS#FractalStructure> <OOVASIS#
    MergerStructure> <OOVASIS#StableSuperStructure> <OOVASIS#
    StrategicAllianceStructure>)
DisjointClasses(<OOVASIS#AmoebaStructure> <OOVASIS#TeamBasedStructure> <
    OOVASIS#VirtualStructure>)
DisjointClasses(<OOVASIS#BioteamingOrganization> <OOVASIS#
    EmpoweredOrganization> <OOVASIS#HypertextOrganization> <OOVASIS#
    LearningOrganization> <OOVASIS#OpenOrganization> <OOVASIS#
    PlatformOrganization> <OOVASIS#ShamrockOrganization> <OOVASIS#
    StrategicOrganization>)
DisjointClasses(<OOVASIS#BusinessProcessReengineering> <OOVASIS#
    CommunitiesOfPractice> <OOVASIS#ComplexAnalyticalMethod> <OOVASIS#
    Kaizen> <OOVASIS#LeanManagement> <OOVASIS#OrganizationalMemory> <
    OOVASIS#SixSigma> <OOVASIS#TaguchiMethod> <OOVASIS#
    TotalQualityManagement>)
DisjointClasses(<OOVASIS#CultureRelation> <OOVASIS#ProcessRelation> <
    OOVASIS#StrategyRelation> <OOVASIS#StructuralRelation>)
DisjointClasses(<OOVASIS#CustomerOrientedStructure> <OOVASIS#
    ProductDivisionalStructure> <OOVASIS#TeritorialStructure>)
DisjointClasses(<OOVASIS#DivisionalStructure> <OOVASIS#
    FunctionalStructure> <OOVASIS#MatrixStructure> <OOVASIS#
    ProjectOrientedStructure> <OOVASIS#TensorStructure>)
DisjointClasses(<OOVASIS#DynamicNetworkStructure> <OOVASIS#
    FishnetStructure> <OOVASIS#InfiniteFlatHierarchyStructure> <OOVASIS#
```

```
    InternalMarketStructure> <OOVASIS#StaticNetworkStructure>)
DisjointClasses(<OOVASIS#HeterarchicalStructure> <OOVASIS#
    HierarchicalStructure> <OOVASIS#HybridStructure> <OOVASIS#
    SuperStructure>)
)
```

Curriculum Vitae

Bogdan Okreša Đurić was born on 2 February 1989 in the city of Smederevo, Serbia. Since his young years, he has been living in Varaždin, Croatia, where he attended elementary and high school. His Bachelor thesis on the topic of database integrity marked the end of his Bachelor studies Information Systems in year 2010 at the Faculty of Organization and Informatics at the University of Zagreb. At the same university he finished Master studies Databases and Knowledge Bases in year 2013 under the mentorship of Markus Schatten, with the thesis on the topic of semantic modelling of business rules. Recognising the value of various opportunities, he used international mobility to study at Karl Franzens University in Graz, and fulfil his internship obligations at Jožef Stefan Institute in Ljubljana and Elettra Sincrotrone in Trieste. After starting his doctoral studies in 2015, as a part of Large-Scale Multi-Agent Modelling of Massively On-Line Role-Playing Games in Artificial Intelligence laboratory at the same University following an early start in publications during his Master studies, he attended as author and delivered oral presentations at international and national conferences and a research stay at the Politechnic University of Valencia. His fields of interest in the context of research are various areas of artificial intelligence, such as multiagent systems, semantic modelling, social network analysis, and computer games. Along with the successful academic career, he is an active member of the local and international society, with a long record of volunteering and active youth work.

Bogdan Okreša Đurić

Ivana Kukuljevića 20
42000 Varaždin
Croatia
+385 91 8856676
dokresa@foi.hr

"Little by little, one travels far" - J.R.R.Tolkien

Education

Doctoral Studies in Information Sciences, *Faculty of Organization and Informatics, University of Zagreb*, Varaždin. 2015–ongoing
Working on ModelMMORPG project, I continued my academic career under the supervision of my mentor Markus Schatten, PhD, with research interests in multiagent systems, agent-based modelling, semantic modelling, knowledge management, social network analysis, etc.

Research Stay, *Politechnic University of Valencia*, Valencia, Spain. 11/2016–02/2017
Scientific training opportunity with Vicente Julian Inglada, PhD, as mentor, and other members of Intelligence Research Group of UPV. I further improved collaboration, continued working on my research, and took two courses.

Master of Informatics, *Faculty of Organization and Informatics, University of Zagreb*, Varaždin, *GPA 4.504*. 2010–2013
Awarded with Dean's Award for humanitarian activities, and for excellence in work in Student Council. Awarded a Special Rector's Award for assisting in the organisation of International Student Research Symposium.

ERASMUS Exchange Student, *School of Business, Economics and Social Sciences, Karl-Franzens University of Graz*, Graz, Austria. 02/2011–06/2011

Bachelor of Science in Information Technology with Distinction, *Faculty of Organization and Informatics, University of Zagreb*, Varaždin, *GPA 4.310*. 2007–2010
Bachelor Thesis titled Database Integrity, mentored by Mirko Maleković, PhD.

Master Thesis

Title: Semantic Modeling of Business Rules

Supervisor: Markus Schatten, PhD

Short description: Some possibilities of semantic modelling of business rules, forming basis of business systems, are shown. Ontology intertwined with business rules allows for a different approach to business applications. Used standards, including RuleSpeak, OWL, SWRL, RIF, UML, OCL, and ORM, ensured an up-to-date content.

Experience

Vocational

TEACHING ASSISTANT, *Artificial Intelligence Laboratory, Faculty of Organization and Informatics, University of Zagreb*, Varaždin. 01/2017–ongoing

Continuing my work at ModelMMORPG project as a doctoral student, with a scientific working title.

Aspects of research:

- large-scale multi-agent systems, and organisational models;
- semantic Web;
- social network analysis.

NOMINAL ASSOCIATE TITLE TEACHING ASSISTANT, *Faculty of Organization and Informatics, University of Zagreb*, Varaždin. 05/2016–ongoing

I am working for Knowledge Managements course.

Detailed achievements:

- developed my teaching skills;
- successfully transferred some of my knowledge.

EXPERT ASSOCIATE IN SCIENCE AND HIGHER EDUCATION, *Artificial Intelligence Laboratory, Faculty of Organization and Informatics, University of Zagreb*, Varaždin. 01/2015–12/2016

I am employed at ModelMMORPG project as a doctoral candidate.

Aspects of research:

- large-scale multi-agent systems, and organizational models;
- semantic Web;
- social network analysis.

BUSINESS ANALYST, *Schiedel proizvodnja dimnjaka*, Novi Golubovec. 09/2014–12/2014

My first full-time job. I was introduced to, and used, SAP BI tool to extract data and create reports for the local and regional management.

Detailed achievements:

- got to know SAP environment, especially SAP BI module;
- worked in team, and assisted colleagues in their everyday and ICT-related problems;
- attended several trainings on SAP BI, and cooperated with regional entities.

ERASMUS+ TRAINEE, *Elettra Sincrotrone Trieste*, Trieste. 05/2014–08/2014

After a call, I was selected to participate in Italo-Croatian Mobility in Europlanning (ICroME) project, as a trainee in Elettra Sincrotrone Trieste.

Detailed achievements:

- development and affirmation of my project management skills;
- worked in a new and challenging environment;
- learned about project funded by the EU.

INTERN, *Jožef Stefan Institute*, Ljubljana. 01/2013–03/2013

See below, similar to ERASMUS Intern.

Student Assistant, *Faculty of Organization and Informatics, University of Zagreb*, Varaždin. 03/2009–01/2013

Noncontinuous. I aided students in their academic assignments, practical classes and courses, namely: Text and Image Formatting, Data Structures, Knowledge-Based Systems, Knowledge Bases and Semantic Web.

Detailed achievements:

- developed my teaching skills;
- helped colleagues achieve course goals;
- worked with diverse people, altering my approach accordingly.

ERASMUS Intern, *Jožef Stefan Institute*, Ljubljana. 03/2012–08/2012

Using ERASMUS programme I was an intern for five months. I worked at the Knowledge Technologies department, using Orange4WS platform, data mining techniques and ClowdFlows platform development, Python, Django, and Orange.

Detailed achievements:

- had my programming skills challenged;
- learned about new technologies;
- worked in a new and multicultural environment;
- broadened my network of people;
- practised teamwork.

Miscellaneous

Part of the V4EYC2021 Team, *Varaždin for European Youth Capital 2021*, Varaždin. 11/2017–ongoing

I am an active member of the team that is working on the Varaždin for European Youth Capital 2021 candidacy project.

Languages

Croatian: Native

English: C1-C2 *Cambridge CAE*

German: B1

Computer Skills

Semantic Web: RDF, RDFS, OWL, SWRL, RIF, SPARQL, XML, Protégé

Office tools & publishing: MS Office, Libre Office, LaTeX, Adobe InDesign

Programming: Python, C, C++, C#, PHP, SQL, HTML, CSS, JS

Project Management: IBM WebSphere Business Modeler, MS Project

Graphics: CorelDRAW, Inkscape, GIMP

General Skills

- by nature friendly, welcoming and communicative to both known and yet-to-be-known people, flourishing in diverse and international environment
- opportunity-welcoming achievement-oriented team-player who can lead, motivate, and innovate
- planning skills, management and leadership skills developed on various occasions
- knowledge acquisition, transfer and utilisation skills trained continuously

Interests

Research: multi-agent systems, semantic web, ontologies, semantic modelling, conceptual modelling, social network analysis, data visualisation, international cooperation

Personal: jazz dance, volunteering, choir singing, international relations, travelling

Academic: research, projects, cooperation, teaching, studying

Volunteering

President, *Youth Council of the City of Varaždin*, Varaždin. 11/2017–ongoing

Member, *Youth Association Varaždin Underground Club*, Varaždin. 11/2015–ongoing

PhD Students' Representative, *Student Council of the Faculty of Organization and Informatics, University of Zagreb*, Varaždin. 10/2015–ongoing

Volunteer and Programme Coordinator, *VAKUUM Club*, Varaždin. 01/2015–02/2017

Performers' Fellow, *Špancirfest*, Varaždin. 08/2016

Performers' Fellow, *Špancirfest*, Varaždin. 08/2015

Various, *Contemporary Dance Days*, Varaždin. 06/2015

Performers' Fellow, *Špancirfest*, Varaždin. 08/2014

Translator, Various, *Triskell*, Trieste. 06/2014

Vice-president and Secretary, *Student Council of the Faculty of Organization and Informatics, University of Zagreb*, Varaždin. 10/2010–06/2013
In almost three years of active service in the Student Council, along with proactive and innovative colleagues, we organised several successful projects, of educational, entertainment, or humanitarian nature. Everything was done free of charge, on voluntary basis.

References

References available per request.

Publications

List of publications available at Croatian Scientific Bibliography, link.

Published Research

[1] M. Konecki, B. Okreša Đurić and L. Milić. 'Using Computer Games as an Aiding Means in Programming Education'. In: *Proceedings of The 5th Multidisciplinary Academic Conference 2015*. Prague, CZ: MAC Prague consulting, 2015, pp. 1–8.

[2] B. Okreša Đurić. 'A Novel Approach to Modelling Distributed Systems: Using Large-Scale Multi-Agent Systems'. In: *Software Project Management for Distributed Computing*. Ed. by Z. Mahmood. 1st ed. Springer International Publishing AG, 2017. Chap. 10, pp. 229–254. ISBN: 978-3-319-54325-3. DOI: 10.1007/978-3-319-54325-3_10.

[3] B. Okreša Đurić. 'Organisational Metamodel for Large-Scale Multi-Agent Systems: First Steps Towards Modelling Organisation Dynamics'. In: *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 6.3 (2017), p. 17. ISSN: 2255-2863. DOI: 10.14201/ADCAIJ2017631727.

[4] B. Okreša Đurić. 'Organizational Metamodel for Large-Scale Multi-Agent Systems'. In: *Trends in Practical Applications of Scalable Multi- Agent Systems, the PAAMS Collection*. Ed. by F. de la Prieta et al. Advances in Intelligent Systems and Computing 473. Seville, ES: Springer International Publishing, 2016. Chap. 8, pp. 387–390. ISBN: 978-3-319-40158-4. DOI: 10.1007/978-3-319-40159-1_36.

[5] B. Okreša Đurić. 'Towards Modelling Organisational Dynamics for Large-Scale Multiagent Systems'. In: *Trends in Cyber-Physical Multi-Agent Systems. The PAAMS Collection - 15th International Conference, PAAMS 2017*. Ed. by F. De la Prieta et al. Advances in Intelligent Systems and Computing 619. Cham: Springer International Publishing, 16th July 2017, pp. 245–248. ISBN: 978-3-319-61578-3. DOI: 10.1007/978-3-319-61578-3_28.

[6] B. Okreša Đurić and M. Konecki. 'Modeling MMORPG Players' Behaviour'. In: *Central European Conference on Information and Intelligent Systems*. Ed. by T. Hunjak, V. Kirinić and M. Konecki. Varaždin, HR: University of Zagreb, Faculty of Organization and Informatics Varaždin, 2015, pp. 177–184.

[7] B. Okreša Đurić and M. Konecki. 'Specific OWL-Based RPG Ontology'. In: *Central European Conference on Information and Intelligent Systems*. Ed. by T. Hunjak, V. Kirinić and M. Konecki. Varaždin, HR: University of Zagreb, Faculty of Organization and Informatics Varaždin, 2015, pp. 185–190.

[8] B. Okreša Đurić and M. Maleković. 'How to Manage Knowledge With Domain Specific and General Conceptual Modelling Examples'. In: *Proceedings of the 19th European Conference on Knowledge Management*. European Conference on Knowledge Management. Ed. by E. Bolisani, E. Di Maria and E. Scarso. Vol. 2. Reading, UK: Academic Conferences and Publishing International Limited, 6th Sept. 2018, pp. 615–622. ISBN: 978-1-911218-95-1.

[9] B. Okreša Đurić and M. Maleković. 'Knowledge Management and Conceptual Modelling Towards Better Business Results'. In: *Proceedings of the ENTRENOVA - ENTerprise REsearch InNOVAtion Conference*. ENTerprise REsearch InNOVAtion Conference. Ed. by M. Milković et al. Split, HR: Udruga za promicanje inovacija i istraživanja u ekonomiji "IRINET", Zagreb, Croatia, 2018, pp. 239–245.

[10] B. Okreša Đurić and M. Schatten. 'Defining Ontology Combining Concepts of Massive Multi-Player Online Role Playing Games and Organization of Large-Scale Multi-Agent Systems'. In: *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Opatija, HR: IEEE, 2016, pp. 1330–1335. ISBN: 978-953-233-086-1. DOI: 10.1109/MIPRO.2016.7522346.

[11] B. Okreša Đurić and M. Schatten. 'Modeling Multiagent Knowledge Systems Based on Implicit Culture'. In: *Central European Conference on Information and Intelligent Systems*. Ed. by T. Hunjak, S. Lovrenčić and I. Tomičić. Varaždin, HR: University of Zagreb, Faculty of Organization and Informatics Varaždin, 2012, pp. 57–61.

[12] B. Okreša Đurić, I. Tomičić and M. Schatten. 'Towards Agent-Based Simulation of Emerging and Large-Scale Social Networks. Examples of the Migrant Crisis and MMORPGs'. In: *European Quarterly of Political Attitudes and Mentalities EQPAM* 5.4 (2016), pp. 1–19.

[13] B. Okreša Đurić, I. Tomičić and S. Vukelić. 'Model Driven Game Quest Scenario Development for Massively Multi-Player Role-Playing Games: A Case Study'. In: *Central European Conference on Information and Intelligent Systems*. Central European Conference on Information and Intelligent Systems. Ed. by V. Strahonja and V. Kirinić. Varaždin, HR: Faculty of Organization and Informatics, University of Zagreb, 2017, pp. 207–212.

[14] B. Okreša Đurić et al. 'MAMbO5: A New Ontology Approach for Modelling and Managing Intelligent Virtual Environments Based on Multi-Agent Systems'. In: *Journal of Ambient Intelligence and Humanized Computing* (12th Oct. 2018). ISSN: 1868-5137, 1868-5145. DOI: 10.1007/s12652-018-1089-4.

[15] M. Schatten and B. Okreša Đurić. 'A Social Network Analysis of a Massively Multi-Player On-Line Role Playing Game'. In: *Proceedings of the 4th International Conference on Modeling and Simulation*. Ed. by B. Kang. Jeju Island, Korea: IEEE, 2015, pp. 37–42. ISBN: 978-1-4673-9828-2. DOI: 10.1109/MAS.2015.19.

[16] M. Schatten and B. Okreša Đurić. 'Social Networks in "The Mana World" - an Analysis of Social Ties in an Open Source MMORPG'. In: *International Journal of Multimedia and Ubiquitous Engineering* 11.3 (2016), pp. 257–272. DOI: 10.14257/ijmue.2016.11.3.25.

[17] M. Schatten, B. Okreša Đurić and I. Tomičić. 'Towards an Application Programming Interface for Automated Testing of Artificial Intelligence Agents in Massively Multi-Player On-Line Role-Playing Games'. In: *Central European Conference on Information and Intelligent Systems*. Central European Conference on Information and Intelligent Systems. Ed. by V. Strahonja and V. Kirinić. Varaždin, HR: Faculty of Organization and Informatics, University of Zagreb, Sept. 2018, pp. 11–15.

[18] M. Schatten, J. Ševa and B. Okreša Đurić. 'An Introduction to Social Semantic Web Mining & Big Data Analytics for Political Attitudes and Mentalities Research'. In: *European Quarterly of Political Attitudes and Mentalities EQPAM* 4.11 (2015), pp. 40–62.

[19] M. Schatten, J. Ševa and B. Okreša Đurić. 'Big Data Analytics and the Social Web - A Tutorial for the Social Scientist'. In: *European Quarterly of Political Attitudes and Mentalities EQPAM* 4.43 (2015), pp. 30–81.

[20] M. Schatten, I. Tomičić and B. Okreša Đurić. 'A Review on Application Domains of Large-Scale Multiagent Systems'. In: *Central European Conference on Information and Intelligent Systems*. Central European Conference on Information and Intelligent Systems. Ed. by V. Strahonja and V. Kirinić. Varaždin, HR: Faculty of Organization and Informatics, University of Zagreb, 2017, pp. 201–206.

[21] M. Schatten, I. Tomičić and B. Okreša Đurić. 'Multi-Agent Modeling Methods for Massivley Multi-Player On-Line Role-Playing Games'. In: *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Ed. by P. Biljanović. Opatija, HR: IEEE, 2015, pp. 1256–1261. ISBN: 978-953-233-082-3. DOI: 10.1109/MIPRO.2015.7160468.

[22] M. Schatten et al. 'Agents as Bots – An Initial Attempt Towards Model-Driven MMORPG Gameplay'. In: *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection*. Ed. by Y. Demazeau et al. Lecture Notes in Artificial Intelligence 10349. Cham, Switzerland: Springer International Publishing, 2017, pp. 246–258. ISBN: 978-3-319-59930-4. DOI: 10.1007/978-3-319-59930-4_20.

[23] M. Schatten et al. 'Automated MMORPG Testing – An Agent-Based Approach'. In: *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection*. Ed. by Y. Demazeau et al. Lecture Notes in Artificial Intelligence 10349. Cham, Switzerland: Springer International Publishing, 2017, pp. 359–363. ISBN: 978-3-319-59930-4. DOI: 10.1007/978-3-319-59930-4_38.

[24] M. Schatten et al. 'Large-Scale Multi-Agent Modelling of Massively Multi-Player On-Line Role-Playing Games – A Summary'. In: *Central European Conference on Information and Intelligent Systems*. Central European Conference on Information and Intelligent Systems. Ed. by V. Strahonja and V. Kirinić. Varaždin, HR: Faculty of Organization and Informatics, University of Zagreb, 2017, pp. 193–200.

[25] M. Schatten et al. 'Towards an Agent-Based Automated Testing Environment for Massively Multi-Player Role Playing Games'. In: *MIPRO 2017 40th Jubilee International Convention Proceedings* (2017), pp. 1361–1366. DOI: 10.23919/MIPRO.2017.7973597.

[26] J. Ševa, B. Okreša Đurić and M. Schatten. 'Visualizing Public Opinion in Croatia Based on Available Social Network Content'. In: *European Quarterly of Political Attitudes and Mentalities EQPAM* 5.1 (2016), pp. 22–35.

[27] I. Tomičić, B. Okreša Đurić and M. Schatten. 'Implementing Agent Roles in Massivley Multi-Player On-Line Role-Playing Games'. In: *Central European Conference on Information and Intelligent Systems*. Central European Conference on Information and Intelligent Systems. Ed. by V. Strahonja and V. Kirinić. Varaždin, HR: Faculty of Organization and Informatics, University of Zagreb, Sept. 2018, pp. 17–21.

[28] I. Tomičić, B. Okreša Đurić and M. Schatten. 'Modeling Smart Self-Sustainable Cities as Large-Scale Agent Organizations in the IoT Environment'. In: *Smart Cities: Development and Governance Frameworks*. Ed. by Z. Mahmood. Computer Communications and Networks. Cham, CH: Springer, 2018, pp. 3–23. ISBN: 978-3-319-76668-3. DOI: 10.1007/978-3-319-76669-0_1.

[29] I. Tomičić et al. 'Self-Sustainable Agent Organizations in Massively Multi-Player On-Line Role-Playing Games – A Conceptual Framework'. In: *Central European Conference on Information and Intelligent Systems*. Central European Conference

on Information and Intelligent Systems. Ed. by T. Hunjak, V. Kirinić and M. Konecki. Varaždin, HR: University of Zagreb, Faculty of Organization and Informatics Varaždin, 2016, pp. 213–217.