Evolvable Accounting Information Systems:

Applying Design Science Methodology and Normalized Systems Theory to Tackle Combinatorial Effects of Multiple GAAP

Els Vanhoof

Departement Accountancy en Financiering
Faculteit Toegepaste Economische Wetenschappen

Promotor: Prof. Dr. Walter Aerts
Co-promotor: Prof. Dr. Jan Verelst

Proefschrift voorgelegd tot het behalen van de graad van
Doctor in de Toegepaste Economische Wetenschappen aan de Universiteit Antwerpen

September, 2016

Universiteit Antwerpen

Acknowledgments

Breaking the tradition of writing acknowledgments, I will write mine in Dutch and I will use a completely different order in which I thank people. I will tell you a story, the story of my life up until now and how my life included writing a PhD.

I wish to start with this beautiful song text from the musical Wicked, to thank everyone: you all have a special place in my life, in my heart!

"I've heard it said,
That people come into our lives
For a reason
Bringing something we must learn.
And we are led to those
Who help us most to grow, if we let them.
And we help them in return.
Well, I don't know if I believe that's true
But I know I'm who I am today
Because I knew you."

Op 24 november 1987 begon mijn verhaal op deze aardbol. En daarvoor moet ik vooral mijn ouders bedanken: zij hebben mij sinds mijn geboorte gesteund in alles wat ik deed, me advies gegeven als ik dat nodig had, geluisterd naar mijn grieven, kortom ik heb een stel super-ouders! (En ik moet mijn moeder ook dankbaar zijn voor alle taalfouten die ze uit dit ganse werk heeft gehaald :)). Bij mijn geboorte kreeg ik een meter en een peter: tante Rachelle en Serge. Ook zij steunden mij in alles wat ik deed en waren er steeds op de belangrijke momenten. Daarnaast heb ik ook altijd kunnen terugvallen op een geweldige familie. Mijn bomma, nu al bijna 12 jaar overleden, maar een super-moderne, lieve, geweldige, spil-van-de-familie bomma. Vava, die al stierf in '99, zwijgzaam, maar altijd een glimlach als je langs kwam. En dan de rest van de familie zoveel om op te noemen, maar allemaal hebben ze net dat iets waarvan je zegt: *ziedet, da's famille!* Bedankt allemaal voor de steun, de hulp! Aussi, la famile française: merci beaucoup pour tout les beaux moments que nous avons eu ensemble! Speciale dank Jesse, Lieze, Kobe (mijn petekindje) en Lore: bedankt om zo'n geweldig lieve nichtjes en neefjes te zijn. Ik ben erg trots dat ik jullie tante Els mag zijn!

Dank aan de geweldige buren! In het speciaal Niels, Jens en Arne met wie ik heel wat samen heb geravot. Bij de buurman en buurvrouw, Carl en Lilianne was ik ook kind aan huis. Al hebben ze nooit goed begrepen dat ik ELS heet en niet Tinneke van Heule...

Een héél belangrijk hoofdstuk uit mijn leven: 't Geels Volkstoneel. Ik ben in die groep geboren, opgegroeid en wie weet zal ik er ooit sterven. Ik heb daar zoveel vrienden gemaakt, zoveel leuke momenten beleefd,... ik kan het niet wegdenken uit mijn leven. Toen ik klein was noemde iedereen mij *Mevrouw de voorzitster*, omdat mijn papa de voorzitter was en iedereen ervan overtuigd was dat ik de boel ging overnemen. Deels kregen ze gelijk, toen ik volwassen werd besloot ik in het bestuur te stappen en na enkele jaren mee te draaien ben ik penningmeester geworden, wat ik met veel plezier doe! Vandaar zijn er een heleboel mensen die ik enorm hard wil bedanken voor alles wat ze voor mij gedaan hebben en voor mij betekenen:

- Mijn twee beste vriendinnen: Hanne en Tinne. Twee dames die ik véél te weinig zie, maar vriendinnen voor het leven. Mensen die je accepteren zoals je bent, die altijd zullen klaarstaan op moeilijke momenten, maar die je ook eerlijk hun mening durven geven. In één adem wil ik ook Frank, de man van Tinne (jaja, ik heb hen gekoppeld!), bedanken. Je bent een super-vriend!
- De bestuursleden, de vaste woensdagploeg en alle andere vaste crews die helpen met het decor, decoratie, techniek, keuken, tappen, culinair, enz... Dat zijn de mensen waarop je echt kan rekenen, die er altijd zullen zijn!
- En dan alle acteurs en regisseurs die ik heb zien komen en gaan. Waarmee ik zelfs een aantal voorstellingen heb mogen maken. Jullie brachten altijd de sfeer in de keet, jullie zorgden voor de vrolijke noot.

Het verhaal gaat verder op de kleuterschool van de Heidehuizen: super-gezellig en oh zo'n lieve Juffen. Dan de lagere school in Ezaart: een klas van 19 die 6 jaar samen zijn gebleven, al kwamen er drie bij over de jaren héén. En wat zijn we allemaal goed terechtgekomen: de basis die daar gelegd is, is ook voor mij de fundering geweest dankzij dewelke ik dit boekje heb kunnen afwerken. Bedankt meester Jos, Juf Rosette, Meester Jef, Juf Myriam, Juf Nele, Juf Betty en meester Jean! Er is één iemand uit dat vroegere klasje, die altijd een speciale plek in mijn hart heeft gehad en ik moet hem bedanken voor onze vriendschap. Wij speelden samen *"Computertje"* en *"Professortje"* en hij daagde mij altijd uit op intellectuele wijze. En toeval of niet... ik schrijf een doctoraat in accounting en beleidsinformatica en mag volgend jaar op de Universiteit les geven... Dus bedankt Wouter, dankzij jou ben in geïnteresseerd en gepassioneerd geraakt door informatica, jij hebt mij geïnspireerd om *"prof"* te worden bedankt! Kleine noot: Wouter behaalde vorig jaar zijn doctoraat in de Biochemie en Biotechnology en is nu lector... Natuurlijk wil ik ook al mijn andere klasgenootjes bedanken, alsook Bo, Yara, Elly en Yusef, uit lagere klassen, voor de mooie jaren samen. Tijdens mijn lagere school periode was ik ook een bezig bijtje: ik heb zwemles gevolgd, turnles en dan uiteindelijk muziekschool en pianoles. Hier heb ik ook vele fijne mensen ontmoet die ik wil bedanken, gewoon om een klein deeltje in mijn leven geweest te zijn.

Op naar het middelbaar... een ware verandering van landschap. Begonnen in de Latijnse, maar toch na 2 jaar *"afgezakt"*, nee ik moet zeggen: geheroriënteerd, naar economie-wiskunde. Hier moet ik ook mijn klasgenootjes en andere vrienden en vriendinnen bedanken voor de vele jaren samen. Ook wil ik graag alle leerkrachten bedanken voor hun inzet. In het bijzonder: Dhr. Guy Delespaul voor zijn inspirerende lessen geschiedenis en godsdienst, Frans Van Uytven voor de muzikale noot, Dhr. Carl Rijmen om mij te introduceren in de wereld van het onderzoek (het archief in London, ons kritisch leren denken en ons iets vernieuwend laten doen. U hebt de basis gelegd voor mijn

onderzoekskwaliteiten... bedankt!), Dhr. Rob Philipsen en Dhr. Kindt om zo'n leuke economie-leerkrachten te zijn en Rob ook voor zijn begeleiding tijdens de stage van mijn lerarenopleiding. En dan... de allerbelangrijkste, meest inspirerende leerkracht allertijden: Den Bobbel! Euh... Dhr. Marcel Peeters. Hij kon lesgeven als geen ander: ik deed 's avonds mijn boek open en hoorde hem zo nog bezig... hier kietelen, stiggie stiggie, zakie zakie, aankopen dan gaan we naar de deur,... Hij heeft echt mijn interesse in economie en meer bepaald in boekhouden aangewakkerd. Hij legde ook altijd alles uit aan de hand van voorbeelden van de *"grote Miet"* en de *"grote Els"*... en zie nu maar: de grote Miet behaalde een doctoraat in Milieueconomie en ik in TEW met een boekhoud-ICT twist.

Ergens onderweg geraakte ik gefascineerd door het dansspel van de eerste revue van het GVT en besloot zelf ook Rock 'n Roll te leren dansen. Jan, een aloude vriend via toneel, zag dat ook wel zitten en we vlogen erin! Bedankt Jan, om een leuke danspartner te zijn, want we hebben vele leuke dingen gedaan ook naast het dansen. Jammer genoeg heb ik er een punt moeten achter zetten toen ik verder ging studeren. Om nog eens in de verf te zetten dat ik altijd een passie voor informatica gehad heb: ik vond het nodig om tijdens mijn middelbare school carrière avondschool informatica te gaan volgen, omdat er in het 5de en 6de middelbaar geen informatica meer in mijn lessenpakket zat. Dan kom ik vervolgens een lief tegen in de donna chatbox, de befaamde Kristof, die verslaafd is aan computers en alles errond. Zo geraak je verknocht natuurlijk.

De keuze voor mijn studies *"Toegepaste Economische Wetenschappen: Handelsingenieur in de Beleidsinformatica"* kwam dus niet als een verrassing. Op de infodagen viel de keuze al snel op Antwerpen: de sfeer, de ontvangst (Prof. Dr. Ann De Schepper is zeer overtuigend!),... het was direct een match. Bovendien was ik niet vergeten wat Dhr. Kindt, de leerkracht economie die uit de privé kwam, had gezegd: om economie te studeren moet je in Antwerpen zijn, die hebben de beste reputatie. Een keuze waar ik nog geen enkel moment spijt van gehad heb! Op kot gaan was evident en ik vond een super-kot bij Wim (behalve die muizenplaag, brrrrr). Daar kom ik dan aan in 't Stad, als enige van mijn middelbare school. Eerst leerde ik Kevin en Dennis kennen. Maar al gauw vond ik aansluiting bij het groepje waar ik de volgende vijf jaar zowat elk groepswerk mee zou maken, altijd samen mee naar de les ging, enz. Met ieder onze eigen insteek hebben we de ganse opleiding tot een goed einde gebracht, lief en leed gedeeld, hard gelachen en soms ook hard gevloekt. Peter, Sebastiaan en Stijn, héél hard bedankt voor die vijf super-leerrijke, maar ook leuke jaren! Daarnaast ook een dankjewel aan al die zovele anderen die vanop een iets verdere afstand tijdens die opleiding ons intellectueel mee uitdaagden: Gilles, Pieter, Mitch, Jan, Johan, Eva, Tim (2x), Kris, Wai Kin, Sultana, William, Wesley en zovele anderen die ons pad kruisten.

Het keerpunt, het moment waarop ik echt wist dat ik wel geïnteresseerd was in onderzoek, was op de geweldige reis naar de USA, georganiseerd door Prof. Dr. De Backer en begeleid door Jan en Dieter. Door met hen te praten op een wat lossere manier dan gewoon op de UA, wist ik gewoon dat dat was wat ik wilde doen: doctoreren. Aan die reis naar de USA houd ik zeer mooie herinneringen over, wat nieuwe vrienden (Stijn en Anouck) en ook mijn zeer goede vriend Kris. Kris, ik kan jou niet genoeg bedanken voor al onze babbels. Jij die mij beter kent dan ik mezelf ken, die me steeds weer doet verbazen dat jij alle puzzelstukjes in elkaar kan doen passen... Bedankt! Ik kan me geen betere vriend wensen dan jou!

De lessen vorderden en vooral die van Prof. Dr. Herwig Mannaert en Prof. Dr. Jan Verelst maakten een diepe indruk: Normalized Systems Theory probeerde ik te doorgronden, met veel aha momenten! Er werd een studentenclub voor de HIBI's opgericht: IMBIT. En ik stond van bij de start paraat om het legal and finance gedeelte te ondersteunen, want alles wat ik kon doen voor mijn geliefde opleiding, deed ik. Ook SID-in's, infodagen enz. Ik was helemaal verkocht! Ik moet iedereen die ooit iets met IMBIT heeft gedaan bedanken, het was geweldig! Speciale dank aan Kim en David voor het super-idee en aan mijn legal and finance team Philip en Hans.

Voor mijn masterthesis koos ik Prof. Dr. Jan Verelst als promotor: een thesis over het toepassen van de Normalized Systems Theory in het boekhouddomein... helemaal mijn ding! Ik werd ook geholpen door Philip en bracht de thesis tot een goed einde. Maar daarmee was ik nog niet aan het doctoreren... Er kwam geen nieuws, dus ik ging op zoek naar een job in de privé en tekende een contract. Maar het kriebelde... ik bleef kijken naar vacatures (en ja, mijn ex, Kristof, had ook liever gehad dat ik op een unief ging werken...) en ja hoor... ik mocht starten als mandaatassistent accountancy op het departement Accountancy en Finaniering. Ik moet toegeven: die eerste ontmoeting met mijn promotor, Prof. Dr. Walter Aerts, was erop of eronder... ik wilde per sé iets met Normalized Systems Theory doen. Walter stond daar wel voor open, maar het was niet helemaal zijn ding. Hup, ik mijn job in de privé afzeggen en op de UA beginnen op 20 september 2010: het begin van een lange, vermoeiende, maar ook zeer mooie reis.

Op die 6 jaar heb ik veel mensen zien komen en gaan... Ik heb veel geleerd, veel leuke dingen gedaan, veel ups en downs en vandaar wil ik mijn collega's héééééél erg bedanken: zij waren mijn van steun en toeverlaat tijdens het schrijven van mijn doctoraat.

- Prof. Dr. Walter Aerts... U hebt mij het vertrouwen gegeven mijn eigen weg uit te gaan en onderzoek te doen in accounting information systems. Het was geen makkelijke reis: ik had veel te leren over onderzoek doen, over schrijven, over les geven enz. Maar uiteindelijk viel alles in zijn plooi: bedankt voor de leerrijke jaren en de leuke momenten!
- Prof. Dr. Jan Verelst... Altijd hebt U mij gesteund in wat ik deed en hebt U mij zo goed mogelijk geholpen mijn weg te vinden. Aanvankelijk vanop de zijlijn, later meer op het voorplan. Dit was echt onmisbaar in dit doctoraatstraject. Bedankt voor de steun en de hulp!
- Prof. Dr. Ann Jorissen, Prof. Dr. Patrick d'Haens en Prof. Dr. Herwig Mannaert: mijn overige interne juryleden. Bedankt om mijn project al die jaren te volgen en te steunen, bedankt ook voor uw hulpvolle suggesties. Daarnaast waren jullie ook fijne collega's: altijd klaar voor een leuke babbel. Ann, bedankt voor de jaarlijkse chocolaatjes-traktatie. Patrick, bedankt voor de fijne samenwerking qua onderwijs. Herwig, bedankt voor de hulp met het prototype.
- Eline, Sanne en Leentje: mijn beste-vriendinnen-op-het-werk. Woorden schieten tekort... jullie zijn altijd mijn steun en toeverlaat geweest. We hebben samen veel meegemaakt, veel geleerd. Bedankt om altijd voor me klaar te staan!
- Beibei, my dear: We started together working for Walter and spend an amazing 4 years together. We shared all the difficulties of Ph.D life (to switch to talking about boys). In the end, you lived with me for 17 days and it was so nice to have YOU around! You'll always be in my heart! Thank you for everything!

- Oveis: je wist dat jou tekstje ook in het Nederlands zou zijn... Maar dit tekstje zal niet voldoen: ik wil meer zeggen, maar dan moet ik een apart boek schrijven :). Oveis, je hebt 4 jaar lief en leed met me gedeeld, in het bijzonder omdat we beide doctoraatsstudenten zijn van Walter... Jou deur stond altijd voor mij open, ook al had je eigenlijk geen tijd. Bedankt voor die onvergetelijke jaren!
- Christine: jij bent altijd zo druk bezig. Toch maakte je tijd: om AlephQ te laten zien of om een babbeltje te slaan, want je vroeg altijd *"Hoe gaat het met jou?"*. Jij kon me steeds weer de moed geven om er terug in te vliegen... Bedankt!
- Peter: de studie-genoot-die-collega-werd, een bescheiden intellectueel die van cola houdt, maar niet van kaas. We brachten al onze tijd aan de UA al samen door. Nu moet ik jou, met pijn in het hart, achter laten. We hebben zoveel samen beleefd, jij hebt me met zoveel dingen geholpen... Je was co-auteur van 2 van mijn papers, stond altijd klaar met goede raad, we waren roomies in Praag... maar je was vooral een zeer goede vriend, je had een luisterend oor en vele wijze woorden. Bedankt voor al die jaren!
- Jeannine, Ingrid, Aline, Sabine en Karin: onze secretaresses en ook een beetje tweede-mama's voor de assistentjes. Altijd stonden ze klaar voor ons, altijd nieuwsgierig en geïnteresseerd in de *"moeilijke"* dingen waar we mee bezig waren, altijd bereid om een babbeltje te slaan. Bedankt!
- Kelly: onze secretaresse voor even, maar niet echt het mama type... jij vrolijkt de boel op: leuke activiteiten organiseren, grapjes uithalen en creatieve dingen verzinnen. Bedankt om zo'n geëngageerde collega te zijn!
- Philip en Gilles: van het *"andere"* departement. Jullie zijn twee top-collega's! Ik heb de eer gehad jullie op congres beter te leren kennen en heb me daar goed geamuseerd. Jullie deur staat altijd open, voor wat hulp (Philip, bedankt voor je hulp met de methodologie en mijn eerste papers), een leuke babbel enz. Bedankt!
- Frank: met jou heb ik zoveel samengewerkt, van jou heb ik veel geleerd. Hoe jij altijd rustig bleef, onder al die stresserende toestanden. Bovendien was je een luisterend oor en had je altijd goede raad. Bedankt!
- Veronique: de vaste waarde op het departement bij de jonge onderzoekers. Jij met je ervaring op alle gebied, die weet hoe ze de dingen moet aanpakken. Je deur stond altijd open voor een vraag of een babbeltje: bedankt!
- Jasmine: de korte tijd dat je bij ons was, heb je toch een grote indruk nagelaten. Het was voor mij geen makkelijke periode en jij was er altijd om te luisteren naar mijn grieven. En om je ongezouten mening te geven! Bedankt!
- Steven: kort en krachtig, dat is jou stijl: Bedankt voor de mooie jaren!
- Kurt: de risico-averse collega die altijd ieders mening vroeg over uiteenlopende onderwerpen. Een aangename collega die altijd voor iedereen klaar stond! Bedankt!
- Robin: stilletjes binnengeslopen, maar niet ongemerkt voorbij gegaan. De Robin-rules zullen me altijd bij blijven. Altijd zeer gevat, altijd goed geluimd. Bedankt!
- Shuyu: the one we never really got to know, because he was always so busy working. Except in Paris: astonished by you deciding to enjoy Paris and immediately starting to ask really personal questions. Thank you for being a pleasant colleague!
- Harry: You have always been busy with things I do not understand (because they are way out of my league). Nevertheless, you were pleasant company to talk about everything that happens in the world. Thank you for the many beautiful years!
- Gertjan: Je brengt leven en sfeer in de keet, maar staat ook open voor een gewone babbel, een vraag of eender wat. Bedankt om zo'n sfeervolle, toffe collega te zijn!

- Tim: staat altijd klaar voor een vrolijke noot of een serieuze, inhoudelijke babbel! Bedankt!
- De vele fijne collega's die ik heb moeten zien vertrekken... Machteld die van aanpakken wist, Andy die me altijd kon doen kalmeren, Jonas die er altijd was voor een leuke activiteit, Lansen een mysterie dat je moest ontdekken, Andriy met ludieke en straffe verhalen... Kim, Dieter, Kris, Jan en Yannis van andere departementen, Niki, Séverine en Ilse: die maar heel even bij ons werkten. En de anderen die ik waarschijnlijk vergeet... Bedankt allemaal om zo'n fijne collega's te zijn!
- Next I wish to thank all foreign Ph.D students and visiting scholars: for brightening up my days, for the interesting conversations, for just being so nice people to be around... Thank you all: Orn, Livia, Parichart, Yan, Lihong, Dalina, Hau, Fabiola, Tran, Rafaël, Yixia, Jie, Poppy, Thrin, Daniel, Benazeer, Kamiel and I hope I did not forget anyone...
- De rest van het professorenkorps vanop gang B3 wil ik bij deze ook bedanken. Met hen had ik interessante gesprekken, van hen heb ik veel bijgeleerd... Bedankt: Jan, Marc, Marc, Ann (van B4), Eddy, Kris, Carlos, Wim en Steven! Ook dank aan alle gastproffen met wie ik de eer gehad heb samen te werken: Nadine, Tom, Bert, Christophe, Sophie en René. En aan het SCOB-team, die altijd (voor mij) op de achtergrond aanwezig zijn: bedankt!
- Ik wil graag ook de vele collega's van de andere departementen en diensten bedanken, die het leven aan de UA nog net iets aangenamer maakten. Via het Young Research Network kregen we de kans elkaar beter te leren kennen: bedankt allemaal! Nog een speciaal woordje van dank aan de lieve mensen van e-campus, die niet enkel technische ondersteuning boden, maar ook echte vrienden zijn geworden.
- Als laatste wil ik al mijn mede-vertegenwoordigers bedanken voor de leuke momenten in de verschillende bestuursorganen en de informele vergaderingen ervoor en erna. En het was natuurlijk zeer aangenaam al die bestuurders van onze universiteit van dichtbij te leren kennen.

Speciale dank aan mijn twee congres-buddy's op Madeira: Edward en Marien. Jullie waren ongelofelijk leuk gezelschap en ik heb veel van jullie geleerd! Also a special thanks to the entire EEWC community and AIS community for their interesting comments and suggestions on the many conferences I visisted! In het bijzonder, Hans Mulder en Jan Hoogervorst, omdat jullie altijd zo vriendelijk zijn en zulke goede ideeën voorstellen!

Om dit doctoraat te kunnen schrijven, was ik afhankelijk van de goede wil van enkele mensen die dagelijks met de problematiek van multiple GAAP bezig zijn: ik wil al deze bedrijven en hun medewerkers van harte bedanken voor hun medewerking, zonder hen had ik dit doctoraat nooit kunnen schrijven. Daarnaast wil ik ook de mensen van NSX bedanken voor hun hulp met het bouwen van mijn prototype, in het bijzonder Sven, Kris, Arco en Paul.

Tijdens mijn doctoraat heb ik ook niet stil gezeten... Ik ging voor het eerst samenwonen, kocht een eigen appartement, verhuisde terug naar huis, verbrak mijn relatie van 8 jaar, ging alleen wonen, werd syndicus van mijn gebouw, werd penningmeester, verloor 11 kg, kwam er 7 terug bij, volgde de lerarenopleiding, ging terug samenwonen, probeerde start 2 run en faalde... Snap je nu waarom die super-collega's zo belangrijk waren?

Op de trein vanuit Aarschot naar Antwerpen in mijn eerste jaar, leerde ik enkele bijzon-

der mensen kennen: Ines, Hannah en Thomas. Ze werden al snel zeer goede vrienden! Annelies vervoegde het groepje als lief (en ondertussen vrouw) van Thomas en Ines kreeg een kindje, Esther. Bedankt allemaal, jullie zijn geweldig!

Een kort woordje van dank voor iemand zeer speciaal die me door een zeer moeilijke periode heeft geholpen: bedankt!

Maar uiteindelijk kwam ik hem tegen... op tinder nog wel: de man van mijn dromen, de man van mijn leven! Lieve schattebollewoefie, lieve Thomas,... Bedankt om mijn licht te zijn in de duisternis! Jij maakt me gelukkig en blij, geeft me de moed om ervoor te gaan! Bedankt lieve schattie! Ik hou van jou, zielsveel! Jij brengt het beste in mij naar boven!

Dankzij Thomas, kreeg ik er bovendien nog een stel geweldige schoonouders, twee paar super-grootouders en een heleboel vrienden bij. Zij waren al snel niet meer weg te denken uit mijn leven. Bedankt allemaal! Bedankt ook aan de ski-club: jullie waren geweldig gezelschap op mijn werk-ski vakantie en super-lief allemaal toen ik mij toch op de latten waagde en mijn mediale-collaterale band scheurde. Bedankt!

Er zijn nog een heleboel mensen die ik niet vernoemd heb: een heleboel vriendinnen en collega's van mijn mams en hun kroost... Bedankt allemaal om altijd zo begaan te zijn met mij, altijd naar mij te vragen en mij zo hard te steunen. Al mijn vrienden van de lerarenopleiding: bedankt! Met speciale dank aan Chris, Nico, Paul, Patrick, Guy en Steven om leuke groepsgenootjes te zijn. En aan Rob, Jan en Frank voor de begeleiding tijdens mijn stage. Bedankt ook aan mijn buren en de mede-eigenaars van het appartementsgebouw: jullie zijn echt aangename mensen om als buren te hebben! En ik ben nog een héél aantal vrienden vergeten te noemen, ... Bedankt allemaal!

Lastly, I want to thank the two external members of my jury: Dr. Pavel HRuby and Prof. Dr. Mieke Jans. Thank you for your helpful comments and suggestions during the last few months. I am looking forward to further discussions.

Kortom... ik wil jullie allemaal van harte bedanken! Om mij te steunen, om er voor mij te zijn... Het was geen makkelijk traject, maar dankzij jullie hulp en steun heb ik dit tot een goed einde kunnen brengen: ne welgemeende

MERCI!

Veel liefs, vele groetjes

Els

Abstract

Disclosure requirements are a primary regulatory issue for companies. Companies need to externally report financial information for different purposes and according to multiple formats (e.g., taxation, statutory filing, investor relations). In this regard, companies can be obliged to report according to more than one set of Generally Accepted Accounting Principles (GAAP). Moreover, both GAAP and regulatory environment tend to change over time. Companies use Accounting Information Systems (AIS) to facilitate their reporting requirements and these need to cope with changes in/of GAAP. Therefore, evolvability of the AIS, the ease of further development, is a major characteristic of interest for AIS management. In this dissertation we study the evolvability of multiple GAAP AIS by using a mixed method approach of design science and case studies. The research is conducted in three phases (problem identification, design and construct).

In the problem identification phase, we define and circumscribe the problem domain: processing of accounting-related events according to multiple GAAP. The lack of literature about evolvability and multiple GAAP makes our work largely exploratory. We conduct five case studies to study multiple GAAP AIS in practice and document the designs of the case-specific AIS. Next, we use Normalized Systems Theory to evaluate each of the case designs with regard to evolvability. The result of the first research phase is the identification of combinatorial effects (violations of evolvability). In the design phase, we use the documented combinatorial effects in order to develop design principles. We theoretically evaluate the design principles by relating them to prior literature and to the case studies. Finally, in the construct phase, we build a prototype that serves as a proof-of-concept for the design principles. We use Normalized Systems Theory to evaluate the prototype with regard to evolvability.

We elaborate and document that evolvability is a major issue in current multiple GAAP AIS and contribute to a solution for the problem by proposing design principles for more evolvable AIS. The prototype that we construct, shows the relevance of the design principles and the feasibility of building an evolvable multiple GAAP AIS according to the principles from Normalized Systems Theory.

Nederlandse Samenvatting

Rapporteringsvereisten zijn een primaire kwestie voor alle bedrijven. Bedrijven moeten financiële informatie rapporteren aan de buitenwereld voor verschillende doeleinden en in verschillende formaten (bv. belastingen, statutaire neerlegging, investeringsrelaties). Vandaar kunnen bedrijven verplicht worden om volgens meer dan één rapporteringsstandaard deze informatie vrij te geven. Bovendien, zijn zowel deze rapporteringsstandaarden als de regelgevende omgeving waarin een bedrijf opereert, onderhevig aan wijzigingen. Bedrijven gebruiken accountinginformatiesystemen om hun rapporteringsvereisten te faciliteren en deze dienen daarom gevolg te geven aan de wijzigingen in/van de rapporteringsstandaarden. Daarom is evolueerbaarheid, het gemak van toekomstige ontwikkeling, een belangrijke eigenschap voor het beheer van accountinginformatiesystemen. In deze doctoraatsthesis bestuderen we evolueerbaarheid van accountinginformatiesystemen die meerdere rapporteringsstandaarden ondersteunen door het gebruik van een gemengde methodologische aanpak van ontwerpwetenschappen en gevalstudies. Het onderzoek wordt uitgevoerd in drie fases (probleemidentificatie, ontwerp en constructie).

In de probleemidentificatie fase, definiëren en omschrijven we het probleemdomein: het verwerken van boekhoudkundig gerelateerde gebeurtenissen volgens meerdere rapporteringsstandaarden. Het gebrek aan literatuur over evolueerbaarheid en het gebruik van meerdere rapporteringsstandaarden maakt dat ons werk grotendeels exploratief is. We voeren vijf gevalstudies uit om accountinginformatiesystemen die meerdere rapporteringsstandaarden ondersteunen in de praktijk te bestuderen en hun ontwerp te documenteren. Vervolgens gebruiken we de Normalized Systems theorie om de ontwerpen te evalueren wat betreft evolueerbaarheid. Het resultaat van deze eerste onderzoeksfase is het identificeren van combinatorische effecten (schendingen van evolueerbaarheid). In de ontwerpfase, gebruiken we deze gedocumenteerde combinatorische effecten om ontwerpprincipes te ontwikkelen. We evalueren deze ontwerpprincipes theoretisch door ze aan voorafgaande literatuur en de gevalstudies te relateren. In de constructiefase, bouwen we een prototype dat dient als bewijs dat de ontwerpprincipes valabel zijn. We gebruiken de Normalized Systems theorie om de evolueerbaarheid van het prototype te evalueren.

In deze doctoraatsthesis documenteren we het probleem van evolueerbaarheid in accountinginformatiesystemen die meerdere rapporteringsstandaarden ondersteunen. We wijden hierover uit en dragen bij tot een oplossing voor het probleem door ontwerpprincipes voor meer evolueerbare accountinginformatiesystemen voor te stellen. Het gebouwde prototype, toont de relevantie van de ontwerpprincipes en ook de haalbaarheid van het bouwen van een evolueerbaar accountinginformatiesysteem dat meerdere rapporteringsstandaarden ondersteunt volgens de principes van de Normalized Systems theorie aan.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The first chapter of this dissertation is concerned with the general introduction of the research. More specifically we will introduce the research problem and motivation in Section 1.1. Next, we explain the importance of evolvability in the context of Accounting Information Systems in Section 1.2. In Section 1.4 we discuss the set of more specific research questions and the research steps that we undertake to tackle the research problem. Finally, we describe the outline of this dissertation in Section 1.5

1.1 The Issue of Multiple Generally Accepted Accounting Principles

Regulatory requirements are a primary issue for all companies: they need to report financial information to the outside world. This issue is interesting to study, because different **G**enerally **A**ccepted **A**ccounting **P**rinciples (GAAP) exist and there are multiple regulations that specify in which GAAP a company needs to report. Moreover, specific situations and regulations can lead to the need for a company to report financial information according to multiple GAAP. Consequently, such a company needs to process its economic events according to multiple GAAP to be able to comply with reporting requirements. In this context we only consider external reporting and not internal reporting needs like business intelligence or cost accounting.

There are several reasons why companies might need to adhere to multiple GAAP. A first reason is that companies need to comply with local legislation, like local accounting and tax regulation rules. For example, the declining balancing method is allowed as depreciation method in Belgian tax legislation, although using this method might not provide a true and fair view on reality, which is required by accounting regulation. Second, companies belonging to an economic group need to report financial information to the parent company, which imposes in which GAAP subsidiaries need to provide financial information. A third and last reason are specific regulating mechanisms that require additional financial reporting. For example, Basel II(I) for banks and Solvency II for insurance companies.

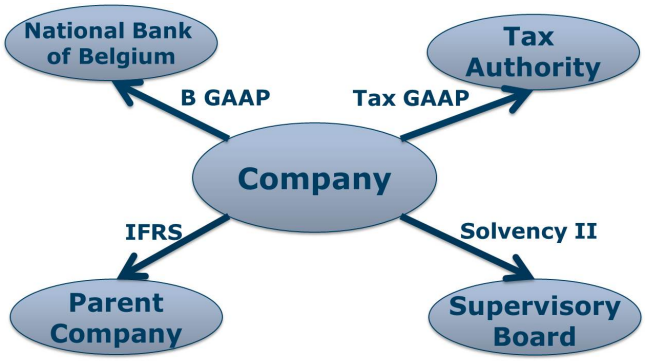

Figure 1.1: Example of an insurance company that needs to adhere to multiple GAAP

We illustrate the need for multiple GAAP reporting with an example of a Belgian insurance company in Figure 1.1. The company needs to adhere to the following GAAP:

- Belgian GAAP[1] for statutory annual financial statements;
- Belgian tax legislation;
- **I**nternational **F**inancial **R**eporting **S**tandards (IFRS) to report to the parent company, that is listed on a stock exchange;
- Solvency II, imposed by the EU for insurance companies.

Different GAAP prescribe different ways to record and process events related to the financial position and performance of a company, which financial information to report and how to present this information (Sinnett and Willis, 2009). There are two kinds of changes we need to take into account: changes in which GAAP a company needs to report its financial information and changes within the GAAP a company needs to comply with.

First, companies over time need to be able to change the GAAP according to which they report. This can be caused by strategic decisions of the company which affect the regulatory environment in which the company operates. For example, mergers and acquisitions might change the GAAP in which the economic group reports and hence influence the reporting GAAP for the individual subsidiaries. Another possibility is additional regulation that is imposed on companies within a certain sector or companies with certain activities. For example, the Solvency II regulation for insurance companies since 2016.

Second, GAAP are subject to rapid expansion, increased complexity and frequent changes (amendments) (Fisher, 2007). Therefore, companies need to adapt their systems to comply with these new GAAP rules. For example, recent events like the credit crisis and corporate fraud scandals have increased the demand for transparency, quality and relevance of financial information. Another change is elaboration in the GAAP of specific issues that were not addressed before. For example, issuance of IFRS 15 (IASB, 2014): a new revenue standard that provides additional guidance on recognition criteria with respect to the timing of recognizing revenue.

[1]We use the term Belgian GAAP to refer to both accounting legislation and additional guidance provided by the Belgian Commission for Accounting Standards.

1.2 Evolvability in Accounting Information Systems

Companies use **I**nformation **T**echnology (IT) to support their business, which makes them heavily dependent on technology and software (Mannaert et al., 2012b; Huysmans, 2011). This is even more true for accounting: accounting relies almost completely on information systems, which makes the work of auditors more difficult, since they need to audit through the information system (Debreceny and Curtis, 2015). Therefore, changes in/of GAAP (as discussed in Section 1.1) also affect the information systems of companies, more specifically the part that handles financial reporting. In the remainder of this dissertation, we will use the broad term **A**ccounting **I**nformation **S**ystems (AIS) that comprises all accounting-related tasks (like financial reporting, management accounting, cost accounting, audit etc.) of the information system of a company.

Evolvability is the measure for the ability of an organization to implement needed changes (Ciraci and van den Broek, 2006; Mäntylä and Lassenius, 2006; Anda, 2007; Mannaert et al., 2012b). Applied in the context of this dissertation, evolvability is the ability to adapt the AIS to changes in and of GAAP. However, different authors use other measures for evolvability, like cost effectiveness (Cook et al., 2000) and *"ease of further development"* (Mäntylä and Lassenius, 2006; Anda, 2007). These measures for evolvability are rather vague and therefore require additional specification. Also in the closely-related agility literature, where agility is the capability of an organization to detect and implement changes, the measures for agility remain vague, for example time, cost, ease, speed, dexterity, efficiency and effectiveness (Fink and Neumann, 2007; Allen and Boynton, 1991; Conboy and Fitzgerald, 2004; Gebauer and Schober, 2006; Mirani and Lederer, 1998; Bajgoric, 2000; Chang and King, 2003; Börjesson and Mathiassen, 2005; Sambamurthy et al., 2003; Weill et al., 2002; Tallon and Pinsonneault, 2011; Lee and Xia, 2010).

Normalized Systems Theory defines an evolvable information system as an information system to which a set of anticipated changes can be applied easily (Mannaert and Verelst, 2009; Mannaert et al., 2011, 2012b). The use of the word *"easy"* in this definition remains vague, but Normalized Systems Theory further specifies the meaning of easy, by using the notion of combinatorial effects: the less combinatorial effects are caused by a change, the more easy it is to change, the more evolvable the company is (Mannaert and Verelst, 2009; Mannaert et al., 2012b). The definition of changing easy then becomes: it is easy to impose changes on the organizational design of a company when the change does not induce combinatorial effects.

Now we define a combinatorial effect: a combinatorial effect is caused when a change that is imposed on a system has an impact that is not only proportional to the nature of the change, but also proportional to the size of the system on which it is imposed (Mannaert and Verelst, 2009; Mannaert et al., 2012b). Applied to the context of multiple GAAP, a combinatorial effect is caused when changes of GAAP rules or complying to a new GAAP has an impact that is not only dependent on the change itself, but is also related to the size of the system. We provide two examples in an accounting context.

The first example is the newly (January 2016) issued IFRS 16 standard about lease accounting which requires to make a distinction between a financial lease and an operating lease for the lessor, but the distinction should no longer be made for lessees. If we consider

only the change for lessees that should account differently for leases now, since no distinction should be made between financial and operating lease, the impact of the change is twofold: 1) the fact that somehow lessees need to change the way they account for leases, 2) all companies that use IFRS need to handle this change. The combinatorial effect is related to the second impact: the fact that the impact of the change is proportional to the number of companies that use IFRS and are lessees, which is a variable that represents the size of the system. In this case, it is not obvious how to avoid this combinatorial effect. Normalized Systems Theory prescribes principles and design patterns that can be followed to avoid combinatorial effects in software but does not provide domain-specific guidance.

For the second example, we consider two companies, A and B, that are completely similar, except company A has an accounting system that accommodates three GAAP, company B's system accommodates only two GAAP. The impact of introducing a new GAAP in the systems is a challenging task for both A and B. However, if we consider the AIS of both companies to be evolvable (in the strict Normalized Systems Theory sense), the change should not be more difficult for company A (that reports in three GAAP) than for company B (that reports in two GAAP). If the impact of the change depends on the number of GAAP the system of the company supports, the change causes one or more combinatorial effects and hence, the system is not evolvable. This would be the case in our example if the impact of the change is larger for company A than for company B. It sounds contra-intuitive that it is more difficult for company A to support an additional GAAP, but that is reality in non-evolvable IT systems: the larger the system, the more combinatorial effects are caused by changes. These combinatorial effects are the cause for the difficulty of applying the change. In Section 2.5 we explain more on the Normalized Systems Theory, why it is relevant and how we use it in this dissertation.

1.3 Research Gap

Regulatory demands are the main driver for changes in the accounting domain that affect AIS. Next to these regulatory demands, more and more accountants are asked to broaden their contributions: reporting on non-financial measures, auditing information systems, implementing management controls within information systems, and providing management consulting services (Grabski et al., 2011). Also, changes in technology are driving changes in the accounting domain (American Accounting Association & American Institute of Certified Public Accountants, 2012; Grabski et al., 2011). An example of such a change is XBRL: because of the development of the XBRL taxonomy, companies now need to use the XBRL format for financial reporting (Grabski et al., 2011).

Although AIS need to cope with these changes and literature suggests research should be done relating to these changes (like for example, business intelligence, decision support systems, business analytics, IFRS, XBRL, security, audit, control,... (Grabski et al., 2011)), there is no research about the design of evolvable AIS. This is odd, since we do find literature that reports on the additional burdens put on companies by regulatory requirements. Some examples are: 1) Pinsker and Li (2008) and Locke and Lowe (2007) discuss the costly implications of XBRL reporting; 2) many companies adopt **E**nterprise

Resource **P**lanning (ERP) systems to comply with the Sarbanes-Oxley Act (Grabski et al., 2011); 3) IFRS reporting (demanding more detailed reporting than most local GAAP) has a significant impact on IT systems (Meall, 2004; Datardina et al., 2011; KPMG, 2008; Steele, 2009).

Several authors acknowledge the need to study the issue of multiple GAAP in AIS, although most research focuses on convergence to IFRS. Grabski et al. (2011) conclude that little research exists about IFRS and ERP systems and that future research should focus on best practices of implementing IFRS in ERP systems and evaluating the impact of IFRS in audit and control procedures. This conclusion is somewhat surprising, since IFRS is not a new issue. Guan et al. (2013) focus on the need to develop a domain ontology about GAAP next to the need for further research about the **R**esources, **E**vents and **A**gents (REA) model. REA provides a shared and validated way of structuring accounting phenomena (in the form of events, resources and agents) (Geerts et al., 2013) to enable the use of the same primary data to derive information for different stakeholders (McCarthy, 1982). We situate this *"recording"* of accounting phenomena in the input phase of Figure 1.2. If recording is done in an innovative way, this primary data can additionally be processed (the process phase of Figure 1.2) into regulatory reporting requirements and business intelligence (Geerts et al., 2013).

Although Figure 1.2 suggests that REA is useful for this second processing phase, the efforts in this area are limited. There is some REA literature about the design of business processes (Geerts and McCarthy, 1999, 2000b, 2001) (to be able to provide a shared enterprise-wide information architecture), however REA literature lacks specification of any *"accounting conclusions"* (Geerts and McCarthy, 2002). This means that they do not consider how the recorded accounting phenomena should be processed and reported according to different GAAP: they do not contemplate journal entries or suggest another structured way to process accounting-related events so the needed information for reporting can be derived (for example, how to calculate the balance of accounts like revenue). Moreover, the recording of events in REA is done independent of the GAAP the company uses, which makes the model powerful, but since limited research is done about the processing of these events, the multiple GAAP issue is an understudied area in design-oriented AIS research.

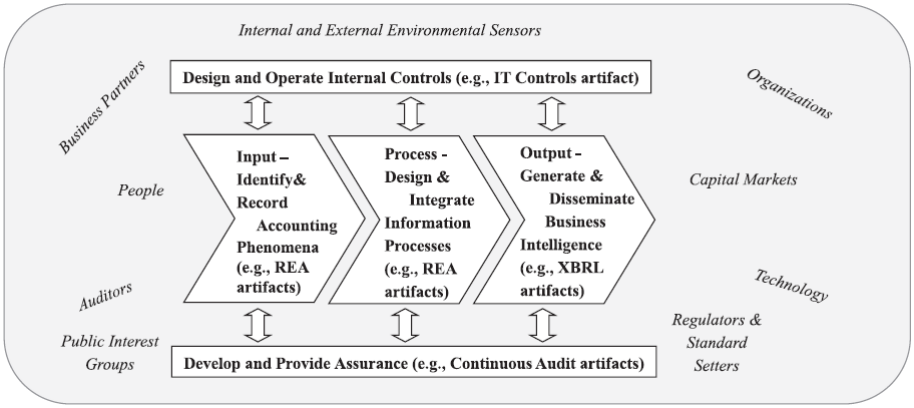

Figure 1.2: Accounting Information Systems Global Environment, adapted from Hunton (2002) (Geerts et al., 2013)

Fischer and Marsh (2012) acknowledge that multinational companies complain that they need to report in multiple GAAP for regulatory purposes, because they find it hard to maintain multiple sets of records. This indicates that AIS are not designed to handle multiple GAAP. Meall (2004) states that it is difficult to adapt AIS to be able to report according to multiple GAAP. Adapting an AIS to comply with IFRS is even more difficult, because, as we already mentioned, IFRS requires a lot more detailed reporting (Meall, 2004; Datardina et al., 2011; KPMG, 2008; Steele, 2009).

One could argue that with the efforts to harmonize GAAP, the issue of compliance with multiple GAAP will disappear over time. However, we believe this is not the case because of multiple reasons. First, convergence of GAAP is still far from reality: GAAP are too different (for example, rules-based US GAAP and principles-based IFRS) and it is not yet clear whether the costs of convergence will outweigh the benefits (Fischer and Marsh, 2012). Second, we see that although in the EU listed companies need to report their consolidated financial statements in IFRS since 2005, currently local regulations are only gradually converging to IFRS. Which implies that for statutory purposes, companies still need to use local GAAP. Third, a risk of convergence towards IFRS is that IFRS might not be relevant/appropriate for all companies all over the world, because of different environmental influences (Fischer and Marsh, 2012). Fourth, we should consider the possibility that harmonization might decrease the quality of reporting since all financial information is provided in only one standard that, like mentioned in our third reason, might not capture the specificity of the company and its environment (Ding et al., 2007).

1.4 From Problem Definition to Research Questions

We have identified the following problems in the previous sections:

- The AIS literature does not study evolvability. We uncover references to evolvability in IT literature, where we find Normalized Systems Theory (which is mainly a software theory) as the most relevant, because it defines evolvability by using the concept of combinatorial effects;
- Multiple GAAP reporting is a burden for some companies, but we do not uncover guidance in literature on how companies can comply with multiple GAAP in an effective and efficient way by using their AIS;
- The changes in GAAP (both which GAAP and changes to the GAAP themselves) require considerable efforts and companies seem not to be able to deal with these changes in an evolvable way.

In this dissertation we use a design science research approach to contribute to a solution for these problems. This is appropriate, because unlike purely empirical research, we want to propose improvements for multiple GAAP AIS with respect to evolvability (March and Smith, 1995). To complement existing research, we do not focus on how to register events, but we focus on how to process accounting-relevant events in multiple GAAP. As we discuss in Section 1.3, REA is mature with regard to its use for the recording activity

(see Figure 1.2), but provides little guidance on processing and reporting of these recorded accounting-related events (In REA an economic event is defined as follows: *"Economic Event represents either an increment or a decrement in the value of economic resources that are under the control of the enterprise."* (Hruby et al., 2006, p. 4)). We propose the following problem domain for this dissertation:

The design of evolvable AIS that support processing of accounting-relevant events in multiple GAAP

We contribute to a solution for the problem by building and evaluating an artifact (Hevner et al., 2004): a prototype of an evolvable AIS that processes accounting-relevant events in multiple GAAP. We use the design science research process of Sonnenberg and vom Brocke (2012) as methodological framework and carry out the first three core activities of the framework (problem identification, design and construct) and their respective evaluation in this dissertation. To design each evaluation activity, we use the **F**ramework for **E**valuation in **D**esign **S**cience research (FEDS) of Venable et al. (2014). In Figure 1.3, we show how the different chapters in this dissertation relate to the design science research process of Sonnenberg and vom Brocke (2012). The remainder of this section addresses the different activities we conduct and which research questions we put forward for each activity.

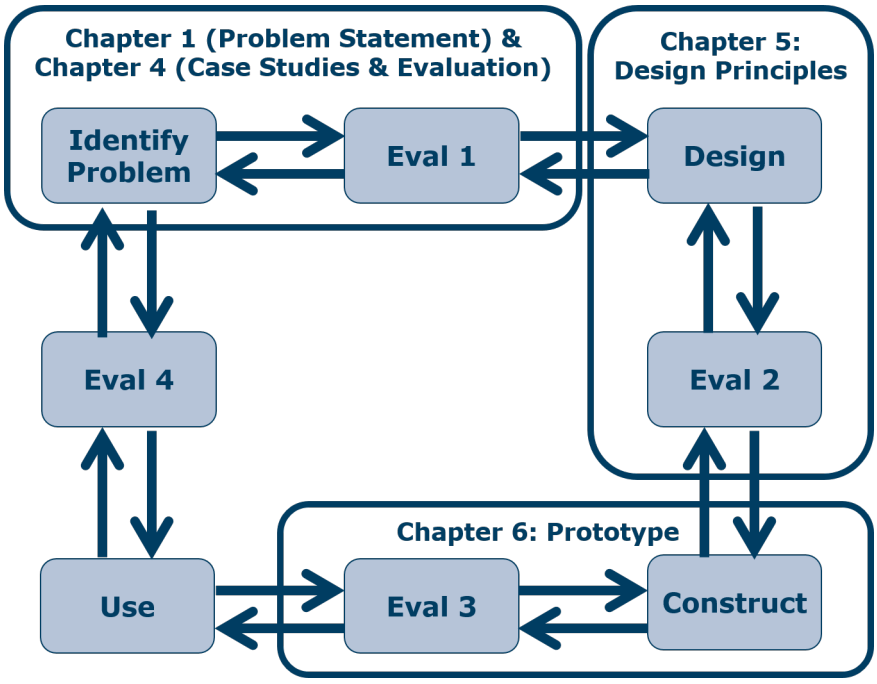

Figure 1.3: Use of the design science research process of Sonnenberg and vom Brocke (2012) in this dissertation

The first activity of the design science research process of Sonnenberg and vom Brocke (2012) is identifying the problem: we situate the problem of evolvable multiple GAAP AIS in Sections 1.1, 1.2 and 1.3 of this chapter. To be able to advance to the second activity, the approach of Sonnenberg and vom Brocke (2012) requires to justify the problem. Because, we find little literature to support the problem statement, we turn to practice: it might be that the reason why we do not uncover literature about evolvable multiple

GAAP AIS is because there is already a good practical solution. Therefore, we perform case studies to study the problem in practice. This approach makes the methodology of this dissertation a mixed method approach (Huysmans and De Bruyn, 2013) (we elaborate on this approach in Section 3.1). We assume that multiple GAAP AIS in practice are effective (with effective we mean that we assume that the AIS serve the purpose they are designed for, being reporting in multiple GAAP) and evaluate whether they are evolvable, by using Normalized Systems Theory. This allows us to structure and analyze the problem. The research question we handle in the first activity hence is:

Research Question 1: Which violations of the evolvability criterion (combinatorial effects) are present in multiple GAAP AIS in practice?

In the AIS designs in the case studies we reveal combinatorial effects. We can justify the problem and research gap of evolvable multiple GAAP AIS, since we find little relevant literature and we have evidence that the problem is not solved (and thus relevant) in practice. In the second activity (design) of the design science research process we develop design specifications for AIS that are evolvable and support multiple GAAP. Using the combinatorial effects from the case studies, we develop design principles that prevent those combinatorial effects in the design of future systems. Therefore, we propose the following research question regarding the second activity:

Research Question 2: Which design principles can be used to design evolvable AIS that can process accounting-related events according to multiple GAAP?

These design principles need to be evaluated for justification. We evaluate the design principles theoretically by relating them to the case studies and Normalized Systems Theory literature. In the third activity (construct) of the design science research process, we build a prototype. This prototype serves as a proof-of-concept for the design principles of the second activity: we test the relevance and the feasibility of the design principles. Therefore, we choose to build a prototype that has a specific, limited amount of functionality, but that provides actual examples. In this way, the design principles are translated into a tangible, concrete design and do not remain vague. However, the resulting prototype cannot be tested in a practical setting because of its limited functionality. Therefore, the research question regarding the third activity is:

Research Question 3: What do we learn from implementing the design principles in a proof-of-concept AIS?

This research question is somewhat general. Therefore, we use more fine-grained sub-questions to scope the research question:

- Can we enforce the design principles?
- If not, how can we redefine them?
- What do we need in addition to the design principles to be able to build evolvable multiple GAAP AIS?
- How feasible is it to build evolvable multiple GAAP AIS?

After building the prototype, we need to evaluate its evolvability. We do this by testing whether the combinatorial effects identified in the problem identification activity still occur in the prototype or not.

1.5 Outline of Dissertation

The design of evolvable multiple GAAP AIS is the problem we identify in this chapter . In Chapter 2, we discuss related literature: the AIS literature (Section 2.1), we elaborate on REA literature (Section 2.2) and we discuss differences in multiple GAAP (Section 2.3). Next, we elaborate on the concept of evolvability in Section 2.4, including reasons to select Normalized Systems Theory as theoretical backbone for this dissertation. Lastly, we provide a more into depth overview of Normalized Systems Theory in Section 2.5.

Next, we discuss the methodological approach in Chapter 3: we use the design science research process of Sonnenberg and vom Brocke (2012) in this dissertation, we use the FEDS of Venable et al. (2014) to design evaluation phases and we discuss how these approaches are related to other design science research literature. We execute three activities from that process in Chapters 4, 5 and 6 (also graphically represented in Figure 1.3).

In the first activity (Chapter 4), problem identification, we first argue the need to study the multiple GAAP problem in practice (Section 4.1). Next, we discuss the case studies in Section 4.2 and the motivation to study SAP extensively in this thesis in Section 4.3. Then we describe the designs of the multiple GAAP AIS from the cases in Section 4.4. The approach of evaluating the designs of the case studies with respect to evolvability is the subject of Section 4.5, the actual evaluation of the designs by identifying combinatorial effects in the designs is discussed in Section 4.6. We conclude in Section 4.7 that multiple GAAP in evolvable AIS is a justified problem to research.

The second activity is the subject of Chapter 5: we develop and evaluate design principles to prevent combinatorial effects. In the introduction of the chapter we discuss our approach. Next, we discuss the development of the design principles: each section describes the manifestation of the combinatorial effect(s), proposes one or more design principles and discusses the evaluation of the principle(s). In Section 5.5 we briefly describe our findings.

Constructing the prototype is the third activity and the subject of Chapter 6. The general approach of building the prototype is described in the introduction of the chapter. In Sections 6.1 to 6.8 we report on the iterative building of the prototype: we develop the prototype incrementally, based on the design principles, propose changes (we use the same changes as in Section 4.6) to the prototype and evaluate the impact of these changes. Next, we illustrate the functionality of the prototype in Section 6.10. In Section 6.11 we briefly describe our findings.

We conclude this dissertation in Chapter 7. Here we first discuss the answers to the research questions in Section 7.1. Then we provide a short conclusion, including our contribution to literature in Section 7.2. Further, we discuss the limitations (Section 7.3), and avenues for further research (Section 7.4).

Chapter 2

Literature Review

In this chapter we discuss literature that is relevant to this dissertation. We start by providing a general overview of the most important topics and methods in AIS literature in Section 2.1. Next, we explain the Resources, Events and Agents model from AIS literature into more detail in Section 2.2, since it is relevant to this dissertation. In Section 2.3 we discuss how GAAP can differ from each other, which is relevant in studying the issue. Since evolvability is an important design goal in this dissertation, we provide an overview of evolvability literature in Section 2.4. Lastly, we discuss Normalized Systems Theory and how we use it in this dissertation in Section 2.5.

2.1 Accounting Information Systems Literature

The AIS literature is multidisciplinary in nature, since it is the discipline that sits between two main disciplines: accounting and auditing on the one hand and computer science and management information systems on the other hand (Debreceny and Curtis, 2015; Debreceny, 2011). Debreceny and Curtis (2015) consider research as AIS related as soon as it can align with the AIS discipline. Main topics in AIS research are: auditing (IT audit and continuous auditing), enterprise systems, accounting systems design, internal controls, IT governance, decision aids (and IT-enabled decision making), information integrity and assurance, internet financial reporting and XBRL, technology adoption and measurement of return on IT investment (Debreceny and Curtis, 2015). Little research is conducted in the following areas: management accounting, small-business and not-for-profit, the impact of AIS on the ethical judgment and behavior (Debreceny and Curtis, 2015). Another classification of AIS topics in order of importance (Ferguson and Seow, 2011): organization and management of an information system, internal control and auditing, judgment and decision-making, capital market, expert systems, artificial intelligence and decision aids, general AIS frameworks, educational issues, the accounting and consulting profession and databases.

In AIS research a wide variety of research methods are used, where empirical and experimental methods are the most popular (Ferguson and Seow, 2011). Model building is also an important research method, but has lost some of its appeal in more recent research in

favor of empirical and experimental methods (Ferguson and Seow, 2011). This is caused by the long-standing debate about balancing empirical methods and design science research (Debreceny, 2011; March and Smith, 1995; Hevner et al., 2004). Before recent work on design science research methodology in information systems (research like Hevner et al. (2004), Gregor and Jones (2007), Peffers et al. (2007) and Gregor and Hevner (2013)) emerged, it was harder to convince an audience of the value of design research (Debreceny, 2011; Geerts, 2011). Nevertheless, in the AIS field, design research has always had its place (Geerts, 2011). Regarding the theoretical framework AIS researchers use, the most popular are theories from economics and cognitive psychology, whereas theories from computer science have been used less and less over the years (Ferguson and Seow, 2011).

However rich the AIS field is, we do not uncover any literature regarding evolvability in AIS. If we specifically look into design literature in AIS, we uncover literature about for example continuous audit (Alles et al., 2006, 2008; Kuhn Jr. and Sutton, 2006) and process mining (Jans et al., 2010, 2011, 2013). However, these designs cannot be used in the context of multiple GAAP reporting. The most relevant design literature in AIS with regard to our problem statement, is the research about the **R**esources - **E**vents - **A**gents (REA) model (McCarthy, 1982; Geerts and McCarthy, 2002). Therefore, we discuss the REA literature in the next section (2.2).

2.2 The Resources, Events and Agents (REA) Literature

The REA accounting model was first introduced by McCarthy (1982) as a way of extending the conventional accounting model by providing opportunities to meet other (management) information needs. According to Geerts and McCarthy (2002), REA can be considered an ontology as it is a specification of a conceptualization (Gruber, 1993) in the sense that it provides a shared vocabulary, taxonomy and definition for objects (and their relations) to be used in the modeling of a specific domain, in this case accounting applications. REA was extended to serve as a theory to design within-enterprise systems (Geerts and McCarthy, 1999) and between-enterprise systems (ISO, 2015). The basic modeling objects or entities are economic events, economic resources and economic agents and are defined as follows:

- *"Economic Event represents either an increment or a decrement in the value of economic resources that are under the control of the enterprise."* (Hruby et al., 2006, p. 4)
- *"Economic Resource is a thing that is scarce, and has utility for economic agents, and is something users of business applications want to plan, monitor, and control."* (Hruby et al., 2006, p. 4)
- *"Economic Agent is an individual or organization capable of having control over economic resources, and transferring or receiving the control to or from other individuals or organizations."* (Hruby et al., 2006, p. 4)

By using this elementary presentation of data, all stakeholders can derive their information from the same primary data (McCarthy, 1982). The resulting model can then be used to design a database which represents an organizations information needs (Geerts, 1993). Over the years REA literature has grown and now REA is presented as an ontology (Geerts and McCarthy, 2000a).

There is still a need to develop the REA ontology further: 1) in the procedural direction, where it is needed that *"accounting conclusions"* (like revenue and profit, we also refer to Section 1.3 and Figure 1.2) are specified, 2) in the declarative direction, where there is a need to fit the REA ontology into more general and more specific, existing frameworks (Geerts and McCarthy, 2002). We can conclude regarding REA that it provides a shared and validated way of structuring accounting phenomena (in the form of events, resources and agents) (Geerts et al., 2013) so it becomes possible to use the same primary data to derive information for different stakeholders (McCarthy, 1982) and this in innovative ways of processing this primary data into business intelligence (Geerts et al., 2013). However, as mentioned above, how accounting conclusions are specified is not yet researched. This dissertation contributes by addressing the processing of events (as defined in the REA model) into information that can be readily reported (like for example the processing of events into journal entries).

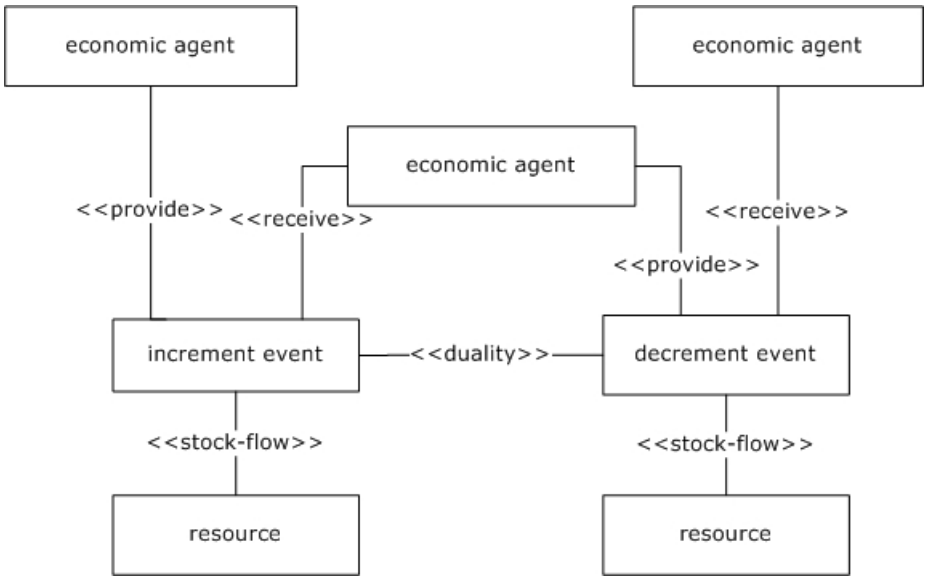

Figure 2.1: The REA accounting model, adapted from Hruby et al. (2006)

Now, we briefly discuss some additional characteristics of the REA model. In Figure 2.1 the REA model as presented by Hruby et al. (2006) is shown. The three basic elements of the REA model are resources, events and agents. Additionally, the model contains relationships between these elements: duality, stock-flow, control (provide and receive in Figure 2.1). The duality relation is the link between two economic events, where one of the events implies a decrease and the other an increase in the resources of the enterprise (Dunn and McCarthy, 1997). This duality relationship can be represented by a process that combines the two economic events (Hruby et al., 2006; Geerts and McCarthy, 2001). There are two kinds of processes: 1) in an exchange process resources are exchanged between parties, 2) in a conversion process resources are used or consumed to produce (or modify) other resources (Hruby et al., 2006).

The stock-flow relationship describes the relationship between economic resources and economic events: production, consumption, use, inflow (also called give) and outflow (also called take) (Geerts and McCarthy, 2000a; Dunn and McCarthy, 1997). The control relationship depicts the relation between the economic agents and the economic events: the provide and receive relationship determines who loses and who receives control over the economic resource as a result of the economic event (Hruby et al., 2006).

Next, there are three restrictions in the REA ontology, presented as axioms (Geerts and McCarthy, 2000a, p. 12):

- *"Axiom 1 – At least one inflow event and one outflow event exist for each economic resource; conversely inflow and outflow events must affect identifiable resources.*
- *Axiom 2 – All events effecting an outflow must be eventually paired in duality relationships with events effecting an inflow and vice-versa.*
- *Axiom 3 – Each exchange needs an instance of both the inside and outside subsets."*

In practice, events that are not (yet) balanced by another event (in a duality relationship) might occur. When this happens, one agent has a claim towards another agent: for example, a purchase has occurred on credit, so a cash disbursement event has not yet taken place (McCarthy, 1982). It is not required by REA to model all claims, but if they are modeled, they are called materialized claims (Hruby et al., 2006). The stock-flow relation between the different processes binds them into a value-chain (or scenario). It provides insight into how an enterprise acquires resources and converts them through different processes into cash. The value chain approach allows building a cascade of models at different levels: from the complete value chain through the task level (Geerts and McCarthy, 1997; Geerts, 1993). The REA methodology advises to identify economic events on the level that decision makers need information to plan, design, follow-up and evaluate activities. To avoid splitting up events gradually, Geerts and McCarthy (2001) suggest an additional and more granular level, the task level.

To represent the task level, system flowcharts and other existing modeling techniques like workflow management and data flow diagrams are used (Geerts and McCarthy, 2001). In addition to creating an event, the task level can be used for the following reasons: when the process is strongly dependent on technology, when the task can only be materialized after the completion of all process tasks or when there is a one to one cardinality between the task and a certain event (Geerts and McCarthy, 2001).

2.3 Multiple GAAP

In this section we discuss how GAAP differ from each other. There are five ways in which the GAAP rules can differ from each other and two additional issues that we should take into account. First, GAAP can differ regarding the definition of concepts, the recognition criteria, the measurement methods, the presentation requirements and disclosure requirements. Second, GAAP alternatives and lack of requirements also influence the way a company handles multiple GAAP. These differences therefore have multiple dimensions which are reflected in the design of the AIS.

First, GAAP can differ in five ways (Fischer and Marsh, 2012):

- Concept definition: basic concepts (assets, equity, liabilities, income and expenses) and definitions used by specific standards (for example, revenue: which sale contracts are covered by the standard and which are not) can differ.
- Recognition criteria: determine if, when and how an item that matches with a definition of an element of financial statements, is recorded in the financial statements. For example, revenue recognition on delivery of goods or when legal title passes.
- Measurement methods: determine the amount included in the financial statements, based on an attribute or model. For example, to record an asset at Fair Value or at historical cost.
- Presentation: differences in presentation style of financial statements regarding terminology, classification, sections to use and type of accounts to use. For example, revenue should be reported separately in the statement of profit or loss.
- Disclosure: differences in additional information to include in the notes to the financial statements and format/depth of these disclosures. For example, report segment information about revenue in the notes.

Second, two additional issues should be taken into account (Fischer and Marsh, 2012):

- Alternatives: GAAP allow alternative recognition and measurement rules (accounting choice). For example, IFRS allows both weighted average and FIFO for inventory measurement.
- Lack of requirements or guidance: when one GAAP alternative does not address an issue that is specifically addressed in another GAAP. For example, IAS 18/IFRS 15 (IASB, 2001, 2014) prescribe revenue recognition criteria, whereas in Belgian GAAP such guidelines do not exist.

2.4 Evolvability literature

The subject of this section is the concept of evolvability. In Section 2.4.1 we provide an overview of what changes drive the accounting domain and how little research is conducted in AIS literature about change. Next we introduce the concepts of agility, evolvability and flexibility from IS literature. In Section 2.4.2 we argue why evolvability is an important IS characteristic. We discuss modularity literature and its shortcomings in Section 2.4.3. Lastly, we argue why we choose Normalized Systems Theory as theoretical framework for this dissertation in Section 2.4.4.

2.4.1 Evolvability and related concepts

The business environment is changing every day and organizations need to be able to respond to it quickly (Harmon, 2014; Tallon and Pinsonneault, 2011; Malhotra and Temponi, 2010; Botha et al., 2008). The volatility of the current business environment in-

creases the need for companies to handle changes, to react to threats and react to new business opportunities (Lu and Ramamurthy, 2011). Changes in a company are reflected in the way the company operates, which we call the organizational design: this can imply changes in the business processes, products, services and/or organizational structures (De Bruyn, 2014). A changing organizational design can have far reaching consequences: we illustrate this further by using accounting examples in the following paragraphs.

Companies use IT to support their business, which makes them heavily dependent on technology and software (Mannaert et al., 2012b; Huysmans, 2011). This is even more true for accounting, which relies almost completely on information systems and hence makes the work of auditors more difficult, since they need to audit through the information system (Debreceny and Curtis, 2015). This implies that changes in the organizational design are reflected in the IT design (De Bruyn, 2014).

We discuss the significance and importance of changes in the context of accounting and more specifically multiple GAAP in Section 1.1 and the lack of research about this in AIS literature in Section 1.3. In the next paragraphs we discuss research from other fields, mainly IT-related, that are useful for our problem domain. In IT, research conclusions can be drawn on different levels: usually three different levels are distinguished being the enterprise level, the business process level and the software level. Terminology in this respect is important, since when the term *"system"* is used, it refers to the entire organization of a company, which includes the IT system, but also the business processes and other organizational aggregations. Therefore, some of the research is conducted at the organizational level, other at the software level or the business process level.

We find relevant research about (organizational) agility, which is the ability of a company to adapt to changes (Neumann, 1994; Fink and Neumann, 2007; Dove, 2002; Weill et al., 2002; Sambamurthy et al., 2003). Agility consists of both detecting needed changes and implementing new solutions to react to the changes (Overby et al., 2006; Sambamurthy et al., 2003). As mentioned above, technology and software are important for organizations and therefore agility of the IT systems is crucial to achieve agility at the organizational level (Breu et al., 2002). Fink and Neumann (2007) summarize different agility definitions from literature by considering three different kinds of IT dependent agility, which they describe as follows:

- IT dependent system agility: *"The ability to accommodate change in information systems without incurring significant penalty in time or cost"* (Fink and Neumann, 2007, p. 442) (based on prior studies by Allen and Boynton (1991), Conboy and Fitzgerald (2004), Gebauer and Schober (2006) and Mirani and Lederer (1998));
- IT dependent information agility: *"The ability to easily accommodate change in the way organizational users access and use information resources"* (Fink and Neumann, 2007, p. 442) (based on prior studies by Bajgoric (2000), Chang and King (2003) and Mirani and Lederer (1998));
- IT dependent strategic agility: *"The ability to respond efficiently and effectively to emerging market opportunities by taking advantage of existing IT capabilities"* (Fink and Neumann, 2007, p. 442) (based on prior studies by Börjesson and Mathiassen (2005), Sambamurthy et al. (2003) and Weill et al. (2002)).

Detection of needed changes is quite straight-forward in the context of multiple GAAP since the main change drivers are regulation and internal changes in the company, which are not difficult to detect. The real issue in multiple GAAP accounting is the actual implementation of these changes, the second part of the agility definition. Therefore, we do not use the term agility in this dissertation, instead we use the term evolvability. Evolvability is the measure for the ability of the organization to implement the needed changes (Ciraci and van den Broek, 2006; Mäntylä and Lassenius, 2006; Anda, 2007; Mannaert et al., 2012b), which only refers to the second part of the agility definition and therefore is more suited in the context of this dissertation.

Another closely related concept is flexibility, which we define as the more stringent property of a system that allows it to change with respect to a pre-set number of parameters that are included in the software design, whereas both evolvability and agility refer to the capability to rapidly reconfigure the software by using new parameters (Bernardes and Hanna, 2009).

To attain agility, evolvability and/or flexibility, literature agrees that good modular design is important (Chung et al., 2005; Baldwin and Clark, 2000; Giachetti et al., 2003; Hoetker et al., 2007; Sanchez and Mahoney, 1996) to prevent that a change or adaption in one module impacts other modules (De Bruyn, 2014; Gershenson et al., 2003; Sanchez and Mahoney, 1996; Parnas, 1972; Schilling, 2000). A survey conducted by Nasir and Iqbal (2008) confirms that the most important characteristics of an evolvable system are design, architecture and encapsultation.

2.4.2 The Importance of Evolvability in System Design

In this section, we argue why evolvability is important when designing software. Agility literature is empirical in nature, starting from the concept of agility and using it to deduct a measure that can be obtained through, for example, a survey. However, this stream of literature does provide us with the measures identified in literature for agile systems: time, cost, ease, efficiency and effectiveness (Fink and Neumann, 2007; Allen and Boynton, 1991; Conboy and Fitzgerald, 2004; Gebauer and Schober, 2006; Mirani and Lederer, 1998; Bajgoric, 2000; Chang and King, 2003; Börjesson and Mathiassen, 2005; Sambamurthy et al., 2003; Weill et al., 2002; Tallon and Pinsonneault, 2011; Lee and Xia, 2010). We find more recent research confirming these measures: Tallon and Pinsonneault (2011) use ease, speed and dexterity in their agility definition and Lee and Xia (2010) efficiency and effectiveness. Therefore, we can conclude that for a system that is agile: 1) changes to the system can be applied easily, 2) it does not cost too much time (it is fast), 3) the costs are relatively low and 4) it can be done in an efficient and/or effective way.

In the more specific evolvability literature, Cook et al. (2000) propose cost effectiveness as a measure for evolvability. Research also suggests to use both structural measures as expert opinions to measure evolvability (Anda, 2007). Other authors like Mäntylä and Lassenius (2006), Anda (2007) and Mannaert et al. (2012b) refer to the *"ease of further development"* in their evolvability definitions. All these measures are rather vague (e.g., an expert opinion) and can only be used to compare systems that have certain similar

characteristics to which the measures are designed. However, these measures reflect the advantages of an agile/evolvable system: they can be adapted faster, at a low price, in an efficient and effective way, easy, etc.

Although agility seems an advantageous characteristic for a system, many studies state that organizational agility in response to the quickly changing environment is problematic (Beer and Nohria, 2000; Hammer and Champy, 1993; Huysmans, 2011). In particular, in software the maintenance cost is frequently identified as one of the most challenging and costly aspects of the life-cycle of software applications (Lientz and Swanson, 1981; Mookerjee, 2005; Rajlich, 2014; Li et al., 2014; Sun et al., 2015; The Standish Group International Incorporated, 2013): research shows that the cost of software maintenance has increased to up to 90% of the total cost of software projects (Rashid et al., 2009). Moreover, we find several (fairly recent) studies that research possibilities to decrease the cost and effort of software maintenance (de Vasconcelos et al., 2016; Midha and Bhattacherjee, 2012; Serna and Serna, 2014). They argue the need for this kind of research because of the high maintenance costs. Since evolvability of a system is the ability to adapt it more quickly, these changes are related to the maintenance of the system in the long run. As maintenance is identified as the most costly domain in software, designing more evolvable systems can significantly reduce maintenance costs.

To explain the lack of evolvability in current systems, we need to investigate the design of software. In this respect three (of the eight) laws of software evolution of Manny Lehman (Lehman, 1980; Lehman and Ramil, 2001) are relevant:

- The law of continuing change: changes need to be imposed on the system to keep the system working in a satisfactory way (Lehman and Ramil, 2001). For example, keep on updating a multiple GAAP AIS to the newest GAAP regulation.
- The law of increasing complexity: if changes are imposed on the system over time, its complexity will increase unless specific action is taken to prevent/reduce it (Lehman and Ramil, 2001). When complexity increases, future changes are more difficult to apply to the system (Prater et al., 2001). For example, when all current and past regulation is added to a multiple GAAP AIS, the system needs to be organized in such a way that one can easily see when which rules are applicable. If this is not done, it is more difficult to change the system with regard to new GAAP regulation, since one should figure out how the rules are applied in the system first.
- The law of continuing growth: a system needs to grow in functional capabilities over time to guarantee user satisfaction over the lifetime of the system (Lehman and Ramil, 2001). For example, when regulation becomes more complex, users will require more assistance from the system to be able to comply with the new rules.

From these laws we learn that any system is going to change and grow and measures should be taken to make sure the complexity of the system does not increase over time (Lehman, 1980; Lehman and Ramil, 2001). Despite the fact that this is not a new issue (Lehman identified them in the 1970's and 1980's), the issue is still not solved as reflected by the increasing costs of software maintenance. Reducing complexity and hence increasing evolvability is mainly studied in modularity literature. The idea of a modular structure dates back to McIlroy (1968), who proposes software as an assembly of modular pieces of software like building blocks. In his view the building blocks can

be used, re-used, changed, upgraded, combined in a simple and evolvable way, without impacting other modules (Gershenson et al., 2003; Sanchez and Mahoney, 1996; Parnas, 1972; Schilling, 2000). Since then a lot of research about modularity has been conducted: about modular structure design, but also about the advantages of good modular design. De Bruyn (2014) provides an overview of some advantages at the modular level (Baldwin and Clark, 2000; Gershenson et al., 2003; Sanchez and Mahoney, 1996; Arnheiter and Harren, 2005):

- **The use of modular operators as proposed by Baldwin and Clark (2000)**: these operators can be applied to modular structures and improve their evolvability as such. For example, substituting an old version of a module by a new one (which is better in performance than the previous one).
- **Complexity reduction**: each module is concerned with one small functionality and does not need to take into account the functionality of other modules.
- **Reusability**: since each module is concerned with a limited functionality, this functionality can be reused more easily than a larger block of different functionalities together.
- **Diagnostability**: communication between modules is very visible and transparent, therefore it is easy to track down problems to the module that caused them.
- **Project management efficiencies**: after determining the functionality of each module, they can be developed separately (even parallel).

These advantages can be translated into economic advantages on the organizational level: because modules can be standardized, they can be easily replaced and even outsourced (Huysmans, 2011). Huysmans (2011) and Van Nuffel (2011) provide an overview of advantages and disadvantages related to an increased organizational performance because of use of modularity:

- **Advantages**
  - *"Simultaneously reconciling flexibility and cost efficiency (e.g., Djelic and Ainamo, 1999).*
  - *Quick adaption to external or internal changes (e.g., Nadler and Tushman, 1999; Sanchez, 1999).*
  - *Ensuring resources decoupling through the reutilization of process specification and best practices (e.g., Worren et al., 2002).*
  - *Increased design options (e.g., Baldwin and Clark, 2000; Mikkola, 2006; Yigit and Allahverdi, 2003).*
  - *Reduce testing cost (e.g., Loch et al., 2001; Sanchez1999).*
  - *Reduced development and production time (e.g., Ulrich, 1995; Yigit and Allahverdi, 2003).*
  - *Enable innovation by substituting modules (e.g., Baldwin and Clark, 1997; Garud and Kumaraswamy, 1995)."* (Huysmans, 2011)
- **Disadvantages**
  - *"Performance trade-off of designing an overmodular structure (Ethiraj and Levinthal, 2004)*

- *Increasing danger of imitation by competitors (Ethiraj et al., 2008)*
- *Modular structures are difficult to design (Baldwin and Clark, 1997)*
- *Competitive dynamics and cognitive complexity limit the applicability of modularity (Ernst, 2005)"* (Huysmans, 2011)

As shown above, modularity of a system can have a number of advantages related to cost-efficiency and flexibility (which is related to evolvability), but as the disadvantages show there is a trade-off between efficiency and modularity and it is difficult to design a modular system. This trade-off is also reflected in the fact that there is no general agreement on how to assess modularity (Gershenson et al., 2003). There are four parameters to describe a modular design problem that each need to be *"appropriate"*: 1) the number of modules, 2) the mapping of design elements to the modules, 3) interactions among the design elements within each module and 4) interfaces or interactions between modules (Van Nuffel, 2011; Ethiraj and Levinthal, 2004). The difficulty and vagueness lies in defining what *"appropriate"* means (Ethiraj and Levinthal, 2004; Gershenson et al., 2003).

We can therefore argue that it is generally accepted that using modularity to attain evolvability leads to cost reductions (both in the development of a system as in maintenance) and other flexibility advantages. In this way we have illustrated the importance and advantages of evolvability (by applying modularity) in the design of systems. However, we also identify possible issues like the difficulty of applying modularity and the costs of a structure that is too modular. In the next section, we first introduce the most important modularity principles from literature and discuss their shortcomings. Later, in Section 2.4.4, we explain why the approach of Normalized Systems Theory is the most useful for the design of modular, evolvable systems.

2.4.3 Modularity Literature

We stated before that to obtain agility and hence evolvability, the use of modularity in the design of a system is essential, therefore, we provide an overview of literature related to modularity. Modularity is used in several scientific domains like for example, computer science, management, engineering and manufacturing (Baldwin and Clark, 1997, 2000). There is not one uniform definition for modularity, but it is generally accepted to associate modularity with *"the process of subdividing or decomposing a system into several subsystems"* (De Bruyn, 2014; Campagnolo and Camuffo, 2010). The use of modularity principles reduces the complexity of a system (Simon, 1962) by decomposing the main problem into smaller sub-problems. Advantages of such a modular design are that it facilitates change (Sanchez and Mahoney, 1996; Baldwin and Clark, 1997, 2000), because the sub-modules can be adapted and replaced easily, which enables re-use (sub-modules can be plugged in easily). To be able to achieve these advantages a module of a system should have certain characteristics which are defined by Baldwin and Clark (2000, p. 63) as follows: a module is *"a unit whose structural elements are powerfully connected among themselves and relatively weakly connected to elements in other units"*.

This definition of a module refers to characteristics that are described by other authors as coupling and cohesion, where they characterize a good modular design with a low degree of inter-modular coupling and a high degree of intra-modular cohesion (Jacobson, 1992;

Ethiraj and Levinthal, 2004). The amount of coupling is proportional to the number of other modules on which the module depends (Abran and Moore, 2004; Orton and Weick, 1990; Page-Jones, 1980). Coupling is related to the interface of the module: the interface is visible to all other modules, the rest of the internal workings of the module is hidden by the principle of information hiding (introduced by Parnas (1972)). The amount of cohesion is proportional to the degree in which the parts of the module are related to one another (Page-Jones, 1980).

Now we discussed the main concepts of modularity, we discuss whether they suffice to actually build evolvable software. The principle of information hiding of Parnas (1972) and further literature about coupling and cohesion is still growing today, elaborating the guidance on how to build a system with low coupling and high cohesion. However, no straightforward answer is given to that question: design patterns exist that decrease coupling and increase cohesion, but these patterns cannot be directly translated into software code, they remain vague and are not sufficient to take clear decisions regarding a specific design or implementation issue (Mannaert and Verelst, 2009; Mannaert et al., 2016). The vagueness of the prescriptive knowledge that exists to design and build evolvable software can also be expressed as a lack of fundamental laws in software design (Porter, 1997; Kruchten, 2005). Moreover, when designers do try to apply the guidance provided, they face time and budget constraints and find it very difficult at a technical level to actually systematically apply the principles (Mannaert et al., 2016).

De Bruyn (2014) summarizes the problem of modularity in a nice way: *"Hence, current designs of modular systems at the software level do not seem to be able to capture all anticipated benefits related to the use of modularity. A theory which prescribes proven and unambiguous principles to guide the design of a good modular software system seems therefore highly desirable within this application domain"*. He only talks about the software level, but later in his thesis he draws a similar conclusion for the organizational level (De Bruyn, 2014): most studies about modularity at the organizational level are merely descriptive (Campagnolo and Camuffo, 2010) and hence research about how to partition designs into modules is missing for organizations and business processes (Ethiraj and Levinthal, 2004; Reijers and Mendling, 2008). Reijers and Mendling (2008) and Reijers et al. (2010) state that there is a need for explicit guidelines regarding modularity in business processes, since currently modularization happens ad-hoc. Also Van Nuffel (2011) acknowledges that both the business process modeling domain (Hull, 2008; Cook et al., 2006; Smith, 2003) and the related service-oriented architecture domain (Arsanjani et al., 2009) suffer from a lack of available deterministic guidelines.

2.4.4 Electing Normalized Systems Theory to Define Evolvability

All mentioned attempts of research have failed to provide straightforward, prescriptive rules to design modular and hence evolvable systems. Therefore, we choose to use Normalized Systems Theory as theoretical framework for evolvable system design: Normalized Systems Theory applies modularity thoroughly and provides a specific measure for evolvability (being absence of combinatorial effects) that can be used (Huysmans, 2011). We believe their approach is the most useful since it is grounded on systems theoretic stability

(Mannaert et al., 2011, 2012b). Other approaches miss the detail (and explicitness) of applying modularity (Huysmans, 2011), lack a specific measure for evolvability that can be used to design evolvable systems (their measures can only be used to compare existing systems) and are tied to a specific context (Mannaert et al., 2011, 2012b). Moreover, Normalized Systems Theory, although a theory initiated at the software level (Mannaert and Verelst, 2009; Mannaert et al., 2012b), has shown its relevance in the design of business processes (Van Nuffel, 2011; De Bruyn et al., 2011; Van Nuffel et al., 2010) and at the enterprise level (Huysmans, 2011). De Bruyn (2014) generalizes the use of Normalized Systems Theory to modular structures in general. This transferability of the basic concepts of Normalized Systems Theory shows its strength for a general use in solving evolvability questions.

Normalized Systems Theory defines an evolvable information system as an information system to which a set of anticipated changes can be applied easily (Mannaert and Verelst, 2009; Mannaert et al., 2011, 2012b). The use of the word *"easy"* in this definition remains vague, but is further explicated by the use of the notion of combinatorial effects: the less combinatorial effects are caused by a change, the more easy it is to change, the more evolvable the company is (Mannaert and Verelst, 2009; Mannaert et al., 2012b). The definition of changing easy then becomes: it is easy to impose changes on the organizational design of a company when the change does not induce combinatorial effects. A combinatorial effect can then be defined as: an effect that is caused when a change that is imposed on a system has an impact that is not only proportional to the nature of the change, but also proportional to the size of the system on which it is imposed. We already provided an example of a combinatorial effect in an AIS context in Section 1.2. In the next section, we will explain Normalized Systems Theory into more depth.

2.5 Normalized Systems Theory

Normalized Systems Theory uses the concepts of stability (from systems theory) and entropy (from thermodynamics) to deduct principles for the design of information systems (Mannaert and Verelst, 2009; Mannaert et al., 2012a). Originally Normalized Systems Theory was applied in software design (Mannaert and Verelst, 2009; Mannaert et al., 2011), but the generalization of the theory by De Bruyn (2014) extends the applicability of the theory to the design of modular structures. A modular structure can be every system that can be split into smaller parts, this can be software, biological systems, physical systems, symbolic systems (book or music) or social systems (De Bruyn, 2014; Simon, 1996).

In the next sections, we will discuss Normalized Systems Theory and how we use it in more depth. To be able to do this, we choose accounting examples and hence the modular structure we identify from our case studies, which we describe into detail in Chapter 4. Therefore, we already include (and repeat) a figure of this modular structure here, in Figure 2.2. We also include and repeat the explanation of this modular structure in Section 4.4.

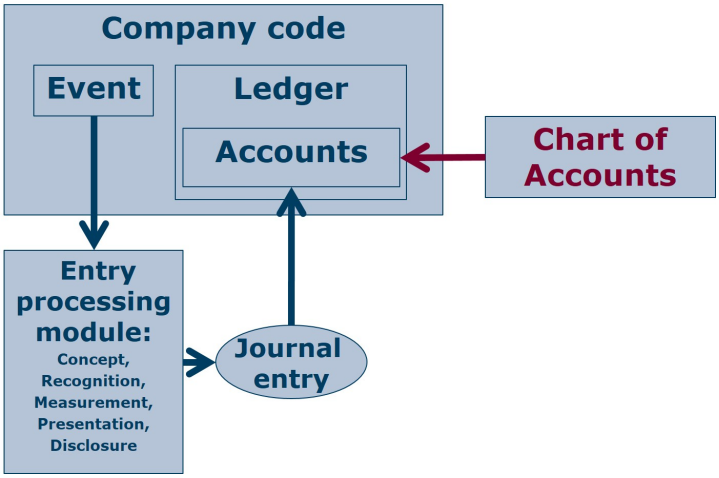

Figure 2.2: Modular structure

2.5.1 Modular Structure in Accounting

By analyzing the case material and consulting the SAP website (SAP AG, 2013) (the use of SAP in this dissertation is explained in Section 4.3), we identify the main components that all AIS from the cases (to be described in Section 4.2) have in common: ledger, accounts, charts of account, entry processing module, journal entry, company code and event. Together these components constitute the modular structure we study, which is generic for reporting in one GAAP and graphically represented in Figure 2.2.

An entry processing module contains five tasks (entry processing tasks) that need to be carried out to determine how to process events, according to the GAAP used. These tasks are: identification and qualification of concepts, recognition criteria, measurement methods, presentation requirements and disclosure requirements. The result of the entry processing tasks is a journal entry, consisting of the accounts that increase and/or decrease and the respective amounts. This implies that we do not consider presentation and disclosure issues that result in other requirements (for example, additional information that needs to be provided in the notes) than posting a journal entry.

Journal entries are posted to a ledger. Next to containing all journal entries, the ledger keeps the totals of each account by adding all amounts posted to the account. The accounts used by the journal entries and collected in the ledger, are hierarchically structured according to a chart of accounts (with an account name and an account number) using a logical categorization of accounts (for example, assets, equity, liabilities, expenses and income).

Company codes are used to separate the financial information of multiple legally separated entities in the same system. Hence, events are always recorded and processed in a specific company code. Therefore, a ledger belongs to one company code. But one company code can hold multiple ledgers. A chart of accounts and an entry processing module can be used by ledgers from multiple company codes. In SAP (SAP AG, 2013) two additional principles have to be applied: 1) within the same company code, all ledgers need to use the same chart of accounts; 2) in order to aggregate financial information from different company codes a separate consolidation module is required.

2.5.2 Primitives, Stability and Entropy

Every modular structure consists of *"primitives"*. This term refers to the building blocks of the studied structure in a certain domain. For example, software consists of primitives that represent data and actions. Data can be defined as the information (attributes) about a certain object (for example, an invoice, an employee,...) that is kept in the information system. An action is an operation/task in an information system that needs to be carried out. The input and output of an action are data (Mannaert and Verelst, 2009). In accounting, we identify the primitives from the modular structure in Figure 4.1: event, entry processing module (with entry processing tasks), journal entry, ledger, company code, accounts and chart of accounts. Another example of primitives in the domain of business processes are the different tasks in a business process, which are the building blocks of the business process.

A modular structure that is stable, is a structure where new changes (which are bounded) can be applied with a bounded impact, even when an unlimited time horizon is considered (i.e. when we assume that the system is subject to new changes over an unlimited time horizon). This bounded impact needs to be interpreted as an impact that is only proportional to the nature of the change and is not proportional to the size of the modular structure it is imposed on. A violation of this stability principle, is called a combinatorial effect. A combinatorial effect hence occurs when a change to a system has an impact that is not only dependent on the change itself, but is also dependent on the size of the system in which the change is applied (Mannaert and Verelst, 2009; Mannaert et al., 2011). A stable modular structure is structured/built in such a way that no combinatorial effects occur when changes are applied, even when the system keeps on evolving/growing indefinitely. In this dissertation we identify combinatorial effects in AIS designs from case studies in Section 4.6 by proposing changes to the designs and evaluating whether these changes cause (a) comabinatorial effect(s).

In order to clarify the concept of a combinatorial effect, let us consider the following example. The modular structure we consider is a multinational company with a number of subsidiaries which each have their own AIS. We assume that all subsidiaries report to the parent using US GAAP. Now assume that the parent wants to use the option of measurement at fair value for financial instruments instead of measurement at historical cost. The impact of the change is proportional to the number of subsidiaries: each subsidiary needs to change its AIS. Since the change is dependent on the size of the modular structure, it causes a combinatorial effect.

In practice we often see that certain changes are not applied, precisely because their impact is so large (the combinatorial effect(s) is (are) large). Assume the same multinational from the last example that wants to change the internal reporting standard from US GAAP to IFRS, because it wants to list on a European stock exchange. The impact of the change is proportional to the number of subsidiaries, hence the change induces a combinatorial effect. This change is considered difficult and would result in a large, costly project in every subsidiary.

Entropy in statistical thermodynamics is a measure for uncertainty in the system (Boltzmann, 1995; De Bruyn, 2014). The measurement of entropy is dependent on the relation-

ship between macro states and micro states of the system. Boltzmann (1995) defines a macro state as the whole of external, observable and measurable properties of the system and a micro state as a set of internal properties of the system that is underlying with respect to the macro state, but is not observable. Entropy then is a function that is dependent on the number of possible micro states that correspond to one macro state in a system (Boltzmann, 1995). Described as a measure for uncertainty in the system: entropy occurs when the same macro state (observable properties of the system) can be caused by different combinations of micro states (not observable), the details of all properties of the systems are unknown and hence it is not certain how the macro state is caused (De Bruyn, 2014). In De Bruyn et al. (2013), the concept of entropy is applied to business processes: entropy increases when detailed information about the process is lost. The macro state then is the state of the whole business process, the micro states are the underlying states of all parts of the business process that determine the macro state. Entropy increases when the macro state can be caused by multiple combinations of micro states, so no information of the parts of the business process is available.

To illustrate entropy, we consider the entry processing module from the modular structure represented in Figure 4.1 as a business process with the entry processing tasks as parts. Now assume that the entry processing module only has one macro state to indicate that something went wrong, namely *"error has occurred"*, then there is no detailed information about the error available. Multiple micro states are possible: the error can be caused in either the concept, recognition, measurement, presentation or disclosure task. Since one macro state (the error) corresponds with multiple possible micro states (error can be caused by all entry processing tasks), entropy is present.

To prevent entropy, a macro state needs to represent the underlying micro states completely and unambiguously (De Bruyn, 2014). Applied to the previous example, the macro state (the error) needs to indicate in which entry processing task the error occurred. We can extend this reasoning at a lower level of detail, saying that we want to know what exactly went wrong in the execution of the entry processing task. For example, information is missing or the calculation could not be executed because of a system overload. In a similar way, you can always refine the level at which the macro state provides information, which means that the uncertainty effect reduces and hence also entropy reduces.

2.5.3 The Theorems from Normalized Systems Theory

To prevent combinatorial and uncertainty effects, the Normalized Systems Theory proposes theorems, derived from the concepts of stability and entropy, to which designers of information systems need to strictly adhere (Mannaert and Verelst, 2009; Mannaert et al., 2012a). We use these theorems to evaluate the design principles in Chapter 5. De Bruyn (2014) generalizes the theorems in order to be able to apply them to modular structures. Therefore we use this generalization to exemplify the theorems. De Bruyn (2014) formulates the Normalized Systems Theory theorems as we describe in the following sections (examples are based on the modular structure from Figure 4.1).

2.5.3.1 Separation of concerns

Theorem: every primitive can only contain one concern (De Bruyn, 2014).

From the stability requirement a concern is a change driver. A change driver is every part of the modular structure that changes independently (Mannaert and Verelst, 2009; Mannaert et al., 2012b). Hence, every primitive can only contain one independent change driver, otherwise stated, separate change drivers need to be separated in a system (De Bruyn, 2014). For example, in a system where two GAAP are used, the GAAP need to be considered as separate change drivers, because they can change independently. This means that an entry processing module cannot depend on more than one GAAP. When the separation of concerns principle is violated, the parts of an entry processing module need to be adjusted every time one of the GAAP changes. In that way new versions of the entry processing module are created in which only one entry processing task is different from previous versions. If a new change needs to be applied to one of the entry processing tasks that occurs in multiple entry processing modules, the impact of the change is equal to the number of duplicated tasks occurring in the different modules. Since the impact is dependent on the size of the system, the change causes a combinatorial effect.

Entropy considers a concern as an information unit. An information unit is a part of the modular structure of which we want to keep information about its execution separate (Mannaert et al., 2012a). Every primitive can only contain one information unit, otherwise stated, information units need to be separated from each other (De Bruyn, 2014). We illustrate this with the example about the error state. If we want information about errors on the level of the entry processing tasks, we consider these tasks as the information units of the system. We need to separate these information units, following the separation of concerns principle. When the principle is violated, an uncertainty effect occurs, since there is only one macro state that states that an error occurred, but it does not portray in which task the error occurred.

2.5.3.2 Separation of states

Theorem: when a primitive uses another primitive in its execution, a state needs to be kept (De Bruyn, 2014).

For example, several measurement entry processing tasks use the Fair Value technique. We identify the primitive entry processing task that uses Fair Value in its execution (next to other tasks that, for example, handle customer discount). A state needs to be kept when executing both primitives, according to the theorem.

If we explain this theorem from a stability point of view, the Fair Value technique needs to keep a state so the entry processing task primitive can retrieve whether the Fair Value is determined or not. This is necessary because if state keeping is not done, the entry processing task needs to be able to handle all possible error states that might occur in the calculation of the Fair Value. It is likely that the Fair Value calculation changes over time: for example, because external information that is needed to be able to determine Fair Value now is automatically obtained through a web service, where it used to be

entered manually. This implies that new errors can occur as a result of this change, for example, the external information is not available or not up to date. If no state keeping is used, the new errors need to be handled by each entry processing task that uses the Fair Value technique: since the impact of this change is proportional to the number of processing tasks that use the Fair Value technique, the impact is proportional to the size of the system, hence the change causes a combinatorial effect. Keeping states after each execution of each entry processing task and task in the execution of the Fair Value technique, prevents this need: the Fair Value technique itself can handle the error situations by defining tasks that need to be executed when a certain error state is reached. The entry processing task using the Fair Value technique can simply observe the state of the Fair Value technique to know whether it is still in progress of determining the Fair Value.

From the entropy point of view, we can explain this theorem by explaining the uncertainty effect in the example above. If no state is kept and the entry processing task receives an error from the Fair Value technique, the entry processing task has no way of knowing what to do with the error, since it has no knowledge of what went wrong. Because the entry processing task has no information about where (in which subtask of the Fair Value technique) the error occurred, an uncertainty effect is present.

2.5.3.3 Version transparency

Theorem: every primitive needs to be able to change without causing incompatibility with other, unchanged primitives that use this first primitive (De Bruyn, 2014).

For example, a GAAP changes its revenue recognition criteria and therefore the used revenue entry processing module needs to be adjusted. Hence, according to this theorem the change cannot cause a change in other primitives like for example, the journal entry. This means that when we assume that a journal entry is a data element, the attributes of the journal entry need to remain unchanged after the change in the recognition criteria for revenue in the entry processing module.

When this principle is violated, a change in the revenue recognition criteria can require a change in all primitives that use it, i.e. all journal entries related to revenue. Since the impact is dependent on the size of the system, the change causes a combinatorial effect.

2.5.3.4 Instance traceability

Theorem: every result of an instance of a primitive needs to be traceable to the specific version and instance of the primitive (De Bruyn, 2014).

An instance is the specific occurrence of a certain primitive from a model. For example in the modular structure we propose, an event occurs in the model. In reality, many different events are recorded by the system: every recorded event in the system is an instantiation of the event from the model.

For example, a GAAP changes its revenue recognition criteria as from January 2016. Therefore the used entry processing module needs to be changed as from this date. After January 2016, the system contains both journal entries posted with the old criteria as with the new ones. The instance traceability theorem demands that every journal entry needs to be traceable with respect to which version of the entry processing module (and hence the recognition criteria) the journal entry is created and which instance of the entry processing module is used.

When this theorem is violated, it is not traceable with which entry processing module and which version of which GAAP a certain journal entry is created. Not being able to trace this, is an uncertainty effect.

2.5.4 Elements as Building Blocks

As adhering to Normalized Systems Theory theorems has proven to be challenging, a set of design patterns have been proposed in certain domains (De Bruyn, 2014). These patterns are called elements and facilitate the application of the theorems in that particular domain. They are the recurring building blocks used to construct the envisioned modular structure. Five elements have been proposed at the software level by Mannaert et al. (2012b): data element, action (task) element, workflow element, connector element and trigger element. As a consequence, the development of a working software application prototype requires the identification of the different instances of each of these elements for the considered domain. Based on the instantiation of these elements, major parts of the actual software code can be generated (each element instance is expanded into a predefined recurring software structure) providing basic out-of-the-box functionalities such as CRUD (create, read, update and delete) screens, waterfall screens, data import, document upload/download, basic user management, basic reporting, etc. Customizations (additional functional requirements which are not offered by the code expansion in a standard way) can afterwards be added in a structured way by software developers, if necessary. In Chapter 6, we use the elements in the construction of the prototype.

2.5.5 Creating Prototypes

Prototypes in Normalized Systems Theory, hence also the prototype in Chapter 6, are built by using the Prime Radiant, a software tool developed by the Normalized Systems Institute (NSI) and Normalized Systems eXpanders factory (NSX). Among other things, this tool provides a graphical user interface (GUI) for the formulation of Normalized Systems Theory software elements. It therefore allows a business analyst to insert the specification of data elements, task elements and flow elements into the Prime Radiant. The analyst can subsequently also expand (generate) and build (compile) software code into a working software prototype. This way, the analyst 1) has a more concrete representation of the envisioned application (a real working application vs. an abstract UML diagram) to validate the completeness, consistency, accuracy, etc. of his model, 2) can use this prototype to communicate and refine the functional requirements in consultation with potential end users (a real working application vs. an abstract UML diagram or

set of use cases), 3) can prove the actual feasibility of the development of a Normalized Systems Theory compatible application of the considered model and domain, and 4) can use this prototype as a starting point to interact with software developers adding custom code into the generated application to provide non-generated functionality (Mannaert et al., 2016). Moreover, it allows an iterative way of working in which a first version of a prototype is built, feedback is given and incorporated in an updated model, after which a new version of the prototype is developed, and so on.

2.5.6 Applying Normalized Systems Theory at the Level of Business Processes

Next to the Normalized Systems Theory theorems, the application of the Theory at the level of business processes and the resulting guidelines and business process design patterns of Van Nuffel (2011) are also relevant for this study. Van Nuffel (2011) proposes guidelines to design evolvable and modular business processes by the use of Normalized Systems Theory. These guidelines are generally applicable, so they also apply in the accounting domain. Discussing all 25 guidelines and the two design patterns proposed by Van Nuffel (2011) would lead us too far from the topic of this dissertation, so we discuss the ones that are relevant to this work: the business rule task and the aggregated business process, which we refer to in Chapter 5 to evaluate the proposed design principles.

The business rule task guideline states that every business rule should be separated in a single task (Van Nuffel, 2011). This guideline is a straightforward application of the separation of concerns theorem: every business rule has a separate change driver that should be separated from other business rules and tasks (Van Nuffel, 2011).

To explain the guideline of an aggregated business process, we first need to explain the guideline of an elementary business process and the related concepts of the definition:

- The guideline of an elementary business process: *"a business process should be operating on one and only one life cycle information object"* (Van Nuffel, 2011; De Bruyn et al., 2014);
- An information object is *"a concrete, identifiable, self-describing entity of information"*, typically an information object *"has an enterprise-wide unique identity, meaningful to a business user and can contain meta-data that describing its data content"* (De Bruyn et al., 2014; Van Nuffel, 2011, p. 343);
- A life cycle information object is the representation of the life cycle of an information object as a business process (Van Nuffel, 2011);
- The elementary life cycle information object (the second guideline of Van Nuffel (2011)) is defined as a non-state transparent information object (Van Nuffel, 2011; De Bruyn et al., 2014).

Following these definitions, we conclude that in order to identify an elementary business process, we should evaluate whether there is a need for the information object to keep a state, because it is used by other primitives in the system, or not. For example (also see Section 2.5.3), the Fair Value technique needs to be state transparent with regard

to the entry processing task (remember that state transparency is always expressed as a relation between two primitives). If we assume no other primitives in the system, the entry processing task could be identified as an elementary life cycle information object. However, if the entry processing task should also exhibit state transparency with regard to, for example, its entry processing module, the entry processing module becomes the elementary life cycle information object. When a life cycle information object is identified, it is necessary to keep information about each task performed on the object (De Bruyn, 2014). Therefore, we need to evaluate whether the state transparent task called by the identified elementary life cycle object can also be identified as another elementary life cycle object. When the state transparent information object is completely incorporated into the life cycle of the primitive using the information object, there is no reason to define it as a separate life cycle information object (Van Nuffel, 2011). For example, a journal entry and a journal entry line object: every action on journal entry lines is originated via the journal entry. But if there are multiple evaluations, business rules and tasks conducted in the same state transparent information object, it is necessary to also identify it as a separate life cycle information object, being a separate business process consisting of several state transitions (Van Nuffel, 2011). This is the case in the example of the Fair Value technique task.

Now, we can explain the guideline of the aggregated business process: it is a business process that only orchestrates several other business processes. An example is an order-to-cash process that needs to be orchestrated, but contains several business processes itself like order entry process, procurement process, production process, delivery process, invoice process (Van Nuffel, 2011). Aggregating multiple elementary business processes can only be justified if there is a need to keep information about the different elementary business processes at an aggregated level (Van Nuffel, 2011).

Chapter 3

Research Methodology

In this dissertation, we use a design science research methodology. The use of design science is mainly motivated by the perceived lack of professional relevance of IS research (Benbasat and Zmud, 1999; Hirschheim and Klein, 2003). We address this by electing a real-world problem: how to design evolvable AIS that are able to process accounting-relevant events in multiple GAAP (March and Smith, 1995; Hevner et al., 2004). Moreover, we design an artifact that solves the problem or improves upon existing solutions, being a prototype of an evolvable AIS that processes accounting-relevant events in multiple GAAP (March and Smith, 1995; Hevner et al., 2004).

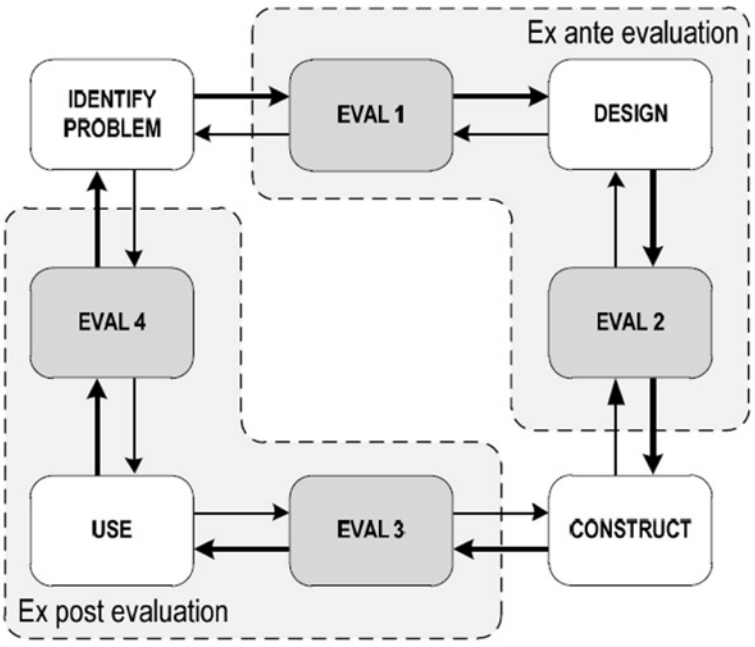

Figure 3.1: The design science research process (Sonnenberg and vom Brocke, 2012)

We use the design science research process as presented by Sonnenberg and vom Brocke (2012) and depicted in Figure 3.1. The advantage of this approach is the focus on evaluation after each stage in the process. Most other approaches put more emphasis on the building of the artifact than on its evaluation (Sein et al., 2011), whereas evaluation

is considered an important part of design science research. Venable et al. (2014) address the issue of the lack of guidance for evaluation in design science research literature by providing a **F**ramework for **E**valuation in **D**esign **S**cience research (FEDS), for designing individual evaluation episodes.

Sonnenberg and vom Brocke (2012) include four activities in their design science research process: problem identification (identify problem), design, construction and use. After each activity they propose an evaluation activity (numbered 1 through 4), which is consistent with the FEDS (Venable et al., 2014). They emphasize that intermediate results can also have a useful contribution to the design science research knowledge base, if they are related to a generic problem and are rigorously evaluated (Sonnenberg and vom Brocke, 2012). This dissertation reports on one design science research process that conducts the first three activities (problem identification, design and construction) and their respective evaluation. We cannot execute the use activity since the prototype of the construct activity is only a proof-of-concept and lacks functionality to be actually used in a practical context, we elaborate on this research limitation in the introduction of Chapter 6. In Table 3.1, we describe the output of each activity, based on the table depicted in Sonnenberg and vom Brocke (2012, p. 393-396). We use these outputs to frame each activity and evaluation phase.

Activity	**Output of the Activity**
Identify Problem	problem statement/observation of a problem, research need, design objectives, design theory and existing solutions to a practical problem
Eval 1	justified problem statement, justified research gap and justified design objectives
Design	design specification, design objectives, stakeholders of the design specification or design tool/design methodology
Eval 2	validated design specification and a justified design tool/methodology
Construct	instance of an artifact (prototype)
Eval 3	validated artifact in an artificial setting (proof of applicability)

Table 3.1: Output of the Design Science Research Process Activities, adapted from Sonnenberg and vom Brocke (2012, Table 3, p. 393-394)

The FEDS of Venable et al. (2014) prescribes into more detail how to evaluate in design science research. The evaluation process consists of four steps: "*1) explicate the goals of the evaluation, 2) choose the evaluation strategy or strategies, 3) determine the properties to evaluate and 4) design the individual evaluation episode(s)*" (Venable et al., 2014). It is especially for step two that additional guidance is provided: the framework requires to answer the questions of why, when, how and what to evaluate (Venable et al., 2014). The why depicts the functional purpose of the evaluation: a formative evaluation is aimed at improvement, a summative evaluation at validation (Venable et al., 2014). Evaluation can take place (the when of the evaluation) before design and construction (ex-ante) or after (ex-post) (Venable et al., 2014). Artificial or naturalistic evaluation refers to the how question of evaluating: if the setting in which the evaluation takes place is a real environment, the evaluation is naturalistic, if not it is artificial (Venable et al., 2014). What to evaluate refers to the properties that are evaluated (Stufflebeam, 2003; Venable et al., 2014).

Next we discuss the three cycle view of Hevner (2007): this paper describes design science as three cycles 1) design cycle, 2) relevance cycle and 3) rigor cycle. In this view the design cycle consists of building and evaluating an artifact, the relevance cycle connects the design cycle with the environment and the rigor cycle connects the design cycle with the knowledge base of scientific foundations, experience, and expertise (Hevner, 2007). The relevance cycle provides feedback from the environment on the design to ensure that the artifact is created with the purpose of addressing a certain problem which is unsolved and important for business (Hevner et al., 2004; Hevner, 2007). The proposed artifact needs to exhibit innovativeness and value, and needs to serve human purposes (March and Smith, 1995). Implicitly Hevner et al. (2004) refer that in this relevance cycle, certain objectives or criteria need to be put forward that address a hitherto unsolved problem, March and Smith (1995) refer to this as evaluation against predetermined criteria. Relevance triggers the research by motivating why a certain design is necessary. After the artifact is built, relevance can be tested by evaluating whether the artifact built is actually useful in the environment.

The rigor cycle ensures the academic value of the research: before starting to design, academic literature should be consulted to identify a research gap (make sure the research is innovative) and relevant design theories and past research in the same domain (Hevner, 2007). When the artifact is built, a contribution to the knowledge base should be made. March and Smith (1995) stress the importance of theorizing (both about the artifact itself and about its environment), because we not only need to determine what works, we must also know why it works. Gregor and Hevner (2013) use the same *"what"* (descriptive knowledge) and *"how"* (prescriptive knowledge), where descriptive knowledge handles phenomena and sense-making, whereas prescriptive knowledge handles about how artifacts should be build (constructs, models, methods, instantiations and design theories). They use the same types of knowledge to describe the research used at the start of building an artifact. The paper of Gregor and Hevner (2013) zooms in on the importance of contributing to knowledge after building an artifact. They acknowledge three levels of contribution types, where level 1 is situating an artifact, level 2 is nacent design theory (knowledge as operational principles or architecture) and level 3 is a well embedded design theory about embedded phenomena.

The design cycle is the core of design science research and most design science literature describes it. In this cycle the artifact is built and evaluated. We interpret the artifact to be build in the broadest sense of its meaning: a construct, a model, a method, an instantiation (March and Smith, 1995; Hevner et al., 2004) or *"new properties of technical, social, and/or informational resources"* (Järvinen, 2007). Regarding evaluation, we already discussed the FEDS (Venable et al. (2014)). Other authors are less clear about evaluation. March and Smith (1995) argue that the artifacts performance should be evaluated with respect to the predetermined criteria. Peffers et al. (2007) agree and add that evaluation can include *"any appropriate empirical evidence or logical proof"*. Hevner et al. (2004) specify that the artifact needs to be evaluated with respect to utility, quality and efficacy in a rigorous way through well-executed methods.

In the next Sections 3.1, 3.2 and 3.3, we discuss the methodological aspects of each activity in the design science research process further. We graphically represent the process and its interaction with its environment (relevance) and the knowledge base (rigor) in Figure 3.2.

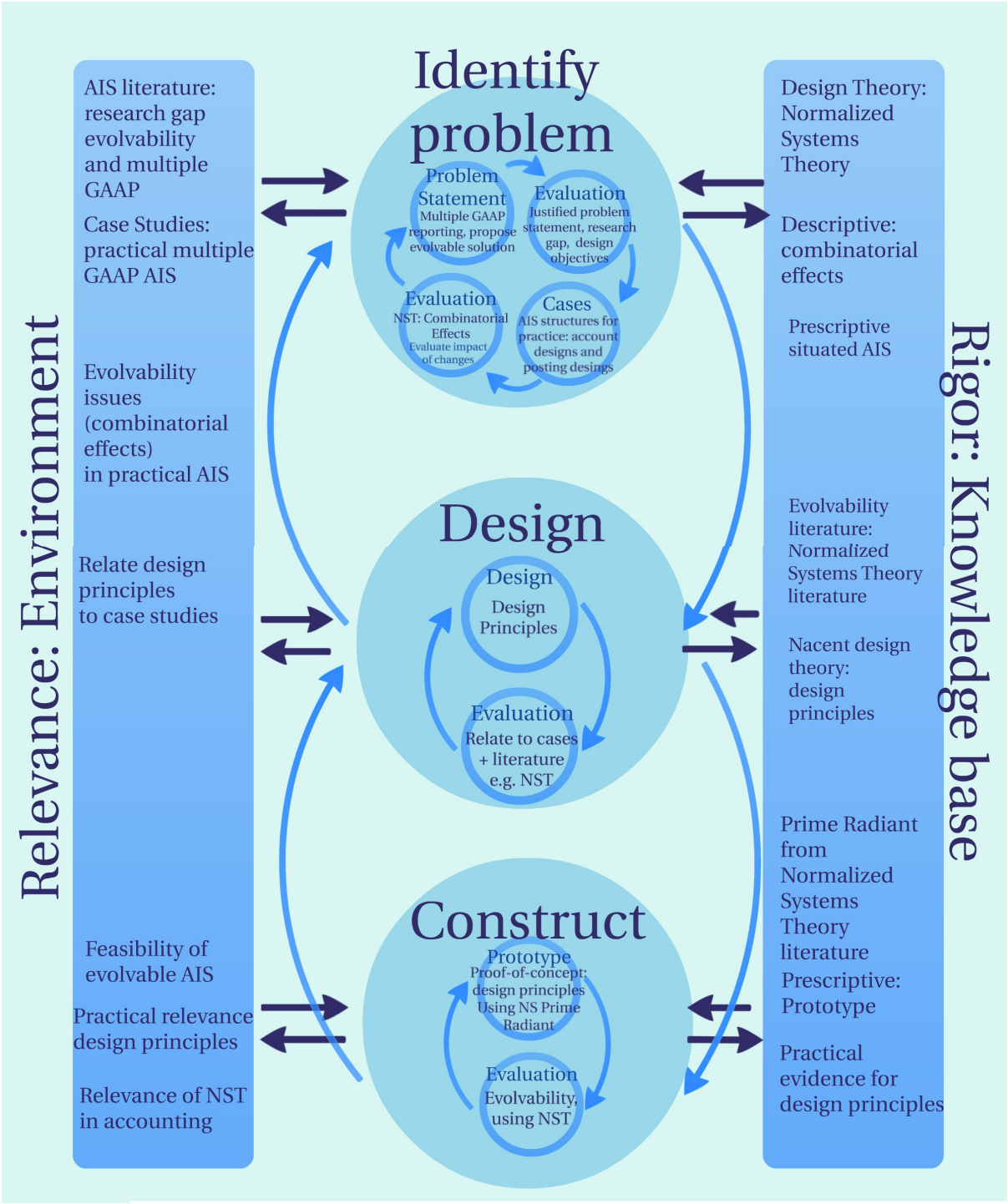

Figure 3.2: The design science research process and its interaction with the environment (relevance) and the knowledge base (rigor)

3.1 First Activity: Problem Identification

In Table 3.1, we mention the output of each activity of the design science research process of Sonnenberg and vom Brocke (2012). For the activity *"identify problem"*, the output is problem statement/observation of a problem, research need, design objectives, design theory and existing solutions to a practical problem; for the activity *"eval 1"* the output is a justified problem statement, justified research and justified design objectives (Sonnenberg and vom Brocke, 2012). In this section we elaborate on how we apply these requirements in the set-up of our first activity.

In the Introduction (Chapter 1), we elaborate on the observation being the multiple GAAP AIS problem, we identify the need for research in this area, we put evolvability forward as the design objective and we propose Normalized Systems Theory as design theory. This provides rigor for this research: we consult relevant literature of the knowledge base (Hevner, 2007) and we use an existing design theory (classified as prescriptive knowledge by Gregor and Hevner (2013)), being Normalized Systems Theory. We find evidence (Meall, 2004; Fischer and Marsh, 2012) that multiple GAAP is an issue for companies and that the main concern for companies are the changing GAAP rules, which makes us believe evolvability is a real problem. Nevertheless, we found the evidence is not convincing enough to start designing improved AIS. Moreover, since there is no design science literature about multiple GAAP AIS or evolvability in accounting, defining the requirements of multiple GAAP AIS seems too difficult. Therefore, we suggest additional research in this problem identification phase: study existing solutions to the multiple GAAP problem to increase the relevance of this dissertation (Hevner, 2007).

Since we use both design science and case studies, we use the mixed methods methodology of Huysmans and De Bruyn (2013) consisting of a design science component and a case study component. This combination is accepted in literature: a case study as part of a design research project (Huysmans and De Bruyn, 2013), in this case during the problem identification phase. Using the classification of Huysmans and De Bruyn (2013), this research design is concurrent exploratory. The theoretical drive of this project, also the dominant component (Huysmans and De Bruyn, 2013), is the design science component. This leaves the case study component as the supplementary component. Regarding pacing of the components, we use a concurrent pacing with the problem identification phase of the design research component as the point of interface.

We use case studies to analyze how companies design AIS to comply with multiple GAAP, we describe the case study methodology in Section 4.2 and the AIS designs from these cases in Section 4.4. Next we use the FEDS framework (Venable et al., 2014) from design science literature to evaluate these designs with regard to evolvability, using Normalized Systems Theory, we describe the evaluation approach in Section 4.5. The result of the evaluation is a list of combinatorial effects (violations of Normalized Systems Theory), which are described in Section 4.6. In Section 4.7 we summarize the results of the case studies and their evaluation. We also judge whether the required output of the first evaluation phase (being justified problem statement, justified research and justified design objectives (Sonnenberg and vom Brocke, 2012)) is sufficient for proceeding with the design and construct of evolvable multiple GAAP AIS.

3.2 Second Activity: Design

After the multiple GAAP problem in evolvable AIS is identified as a valid problem to study using design science, we can proceed to the next step in the design science research process of Sonnenberg and vom Brocke (2012): design. In Table 3.1, we mention the output of each activity of the design science research process of Sonnenberg and vom Brocke (2012). For the activity *"design"*, the output is design specification, design objectives, stakeholders of the design specification or design tool/design methodology; for the activity *"eval 2"* the output is a validated design specification and a justified design tool/methodology (Sonnenberg and vom Brocke, 2012).

After identifying combinatorial effects in AIS from practice, we propose design principles to be used for building future AIS that do not contain these combinatorial effects. These design principles form the design specification (Sonnenberg and vom Brocke, 2012). The way in which we derive these design principles is part of the design methodology (Sonnenberg and vom Brocke, 2012). We describe the details of this methodology in Chapter 5, including the use of the FEDS of Venable et al. (2014).

Our evaluation approach, as described in Chapter 5, ensures both rigor and relevance (Hevner, 2007) of the proposed design principles. We check the relevancy of the design principles by relating them to the case studies and the rigor by relating them to relevant literature. This evaluation provides validation for the design specification, being the design principles. The design principles are a form of prescriptive knowledge and can be considered a level 2 contribution according to the framework of Gregor and Hevner (2013), as nacent design theory.

3.3 Third Activity: Construct

After specifying a design, the construct can be build. In Table 3.1, we mention the output of each activity of the design science research process of Sonnenberg and vom Brocke (2012). For the activity *"construct"*, the output is an instance of an artifact (prototype); for the activity *"eval 3"* the output is a validated artifact in an artificial setting (proof of applicability) (Sonnenberg and vom Brocke, 2012). Hence, in the construct phase we build a prototype of an evolvable multiple GAAP AIS, using the design principles from the design phase and in the evaluation we validate this artifact. We report on these activities in Chapter 6, where we also provide more details on the methodology and the evaluation (using the FEDS of Venable et al. (2014)).

Using the results of Chapters 4 and 5, we ensure the relevance of this third phase. We do not test the prototype in a real live context, which is a possibility for future research. Rigor in constructing this prototype is provided by the use of the Prime Radiant, which allows us to easily build software that adheres to the Normalized Systems Theory theorems. As a contribution to the knowledge base, the prototype classifies as a level 1 contribution and is prescriptive in nature (Gregor and Hevner, 2013).

Chapter 4

Problem Identification: Detecting Evolvability Issues in the Designs of AIS in Practice

4.1 Initial Problem Statement

In the Introduction (Chapter 1) of this dissertation, we elaborately describe the following problems:

- The AIS literature does not study evolvability. We uncover references to evolvability in IT literature, where we find Normalized Systems Theory (which is mainly a software theory) as the most relevant, because it defines evolvability by using the concept of combinatorial effects;
- Multiple GAAP reporting is a burden for some companies, but we do not uncover guidance in literature on how companies can comply with multiple GAAP in an effective and efficient way by using their AIS;
- The changes in GAAP (both which GAAP and changes to the GAAP themselves) require considerable efforts and companies seem not to be able to deal with these changes in an evolvable way.

Therefore, we derive the problem domain: *"The design of evolvable AIS that support processing of accounting-relevant events in multiple GAAP"*. However, we also mention that, although we believe that the lack of literature can justify the problem statement, we do consider it is necessary to investigate the problem further in practice. Notwithstanding literature that calls for research, it might still be possible that evolvability is not an issue in practical multiple GAAP AIS, because a readily practical solution already exists. A second reason to study the problem in practice is to see to what extend there are practical solutions for the problem and to use these solutions as a starting point to further develop multiple GAAP AIS.

To study the problem in practice, we conduct case studies. We explain the methodological approach of the case studies in Section 4.2. Because three out of five case companies use SAP, we refer to SAP frequently. The other two case companies use custom-built software and therefore, we did not study an AIS design from another vendor. Therefore, we emphasize the reasons and rationale of the use of SAP in this dissertation in Section 4.3. In Section 4.4 we describe the multiple GAAP AIS designs from the cases. Next we explain how we will evaluate these designs with regard to their evolvability by using the FEDS from Venable et al. (2014) in Section 4.5. We describe the result of the evaluation, being the identified combinatorial effects in Section 4.6. After this practical study, we conclude in Section 4.7 that it is justified to study the problem of evolvability in multiple GAAP AIS, using design science methodology.

4.2 Case Studies

To gain more insight into the problem of multiple GAAP in practice we use case studies and expert interviews. This is appropriate because we study a contemporary phenomenon (multiple GAAP) in its natural setting to gain deeper insight into the matter (Benbasat et al., 1987; Yin, 2003). Exploratory case studies are appropriate, since we apply a theoretical concept (Normalized Systems Theory) in a new setting (Marshall and Rossman, 2006; Huysmans, 2011).

We choose a collective case study approach and a heterogeneous sample to increase generalizability of the results: identifying combinatorial effects that are inherent in multiple GAAP AIS (Ritchie et al., 2003; Huysmans, 2011). Therefore, we include companies from different sectors: an insurance company, a manufacturer in the graphical and medical sector, a pharmaceutical manufacturer and two transportation firms. The population from which we draw the cases are companies that report according to multiple GAAP. Two case studies cooperate because of personal connections through colleagues. To select additional cases we contact Belgian subsidiaries of listed companies, because they need to prepare statutory annual financial statements using Belgian GAAP and report to the parent company using another GAAP like for example, IFRS. The number of case studies is limited, since we need a high level of detail to be able to analyze the results (Stake, 1994, 1998; Huysmans, 2011). We conduct five case studies: four of them were conducted before the between-case analysis, the fifth case was added after all analysis were already conducted and reported on (even the design principles were already developed). We experienced that the fifth case study only marginally adds to the results (only providing more evidence for conclusions that were already drawn), giving us reasons to believe that this number of case studies suffices, since we reach the point of theoretical saturation (Eisenhardt, 1989). Moreover this number falls within the range of four to ten cases, which is considered to be enough to draw valid conclusions (Eisenhardt, 1989).

We conduct interviews with employees in the financial accounting department and, in two cases, also with key informants from the IT department who are involved in the AIS design. The main focus of the interviews is the functional aspects of the AIS. Interviewing multiple key informants provides us more insight into the specific design of the AIS. The interviews are conducted at each company's site by the author of this dissertation and

last approximately 120 minutes each. During the interviews notes are taken, which are electronically archived. Customary in exploratory research, we use open questions and adapt the questions after each interview (Eisenhardt, 1989) which enables us to gain considerable insight into the existing design of the AIS of the cases. However, we do focus on the processing of events in multiple GAAP in these interviews, since this is the focus of this dissertation.

Next to the case studies, we interview two practitioners: an SAP Belgium representative who is responsible for localization (localization concerns country specific pre-configuration of SAP to comply with local regulation) and someone who works as a business intelligence consultant (for a software company) but permanently works with a client in the banking sector. These interviews help us to gain deeper insight into the problems of multiple GAAP, which allows us to analyze the case studies more thoroughly.

4.3 SAP in this Dissertation

In the remainder of this dissertation we refer to the SAP website (SAP AG, 2013) multiple times and compare our results with the possibilities in SAP . Therefore, it deserves some explanation why we use SAP as the reference ERP system and not the software of other vendors like Oracle, Infor or Microsoft Dynamics AX.

When starting to conduct this research, we did not specifically include or exclude companies that use certain software or not (also see 4.2). Three out of the five case companies we investigate use SAP, the two others use custom-build software. To explain this, we look at figures of market shares of the different ERP vendors in Belgium: we summarize them in Table 4.1. Although we are not completely convinced that these numbers are correct (a drop from a 80% market share to a 21% market share in one year seems strange), they do provide an indication of the ERP landscape in Belgium with SAP as the market leader in the segment of companies over 500 employees. In 2016 the ICT yearbook (Stoffels , 2016) does not report numbers for Oracle anymore, but puts more emphasis on custom-made ERP software. Other sources confirm SAP as market leader in Belgium with 29% of organizations with more than 50 employees that use an ERP solution from SAP and a 69% market share in the segment of multinationals (figures from March 2016 by Computer Profile (2016)). Moreover, we find figures that also confirm SAP as market leader worldwide with a market share dropping from 26% in 2013 to 23% in 2015 (Panorama Consulting Services, 2015; Columbus, 2014). Since our research exactly targets multinational, large companies (because those are the companies that need to handle multiple GAAP), it is not surprising that three out of five of our case companies use SAP.

There are four reasons why we believe it is reasonable to base ourselves mainly on SAP (and the documentation on its website: SAP AG (2013)). First, three of our five case studies use SAP, which obliges us to refer to SAP multiple times. This also shows that SAP has an important position in the market of companies that report in multiple GAAP. Second, as we discuss in the first paragraph of this section, SAP has a market leader position, which is the most apparent in the segment of large companies (>500

Segment	Year	SAP	Oracle	Microsoft	Custom made	Other
<50 employees	2016	10%	n.a.	14%	13%	63%
	2015	8%	6%	18%	n.a.	68%
	2014	6%	6%	23%	n.a.	65%
	2013	11%	6%	28%	n.a.	55%
	2012	4%	4%	50%	n.a.	40%
50-500 employees	2016	11%	n.a.	16%	13%	60%
	2015	14%	8%	17%	n.a.	61%
	2014	14%	7%	13%	n.a.	66%
	2013	42%	19%	4%	n.a.	35%
	2012	16%	10%	42%	n.a.	32%
>500 employees	2016	17%	n.a.	12%	12%	59%
	2015	23%	9%	12%	n.a.	56%
	2014	21%	12%	15%	n.a.	52%
	2013	80%	12%	4%	n.a.	4%
	2012	69%	9%	0%	n.a.	22%

Table 4.1: Market share of ERP vendors in the Belgian market, own composition based on the ICT yearbooks: Smart Business Editiorial Office (2012), Visterin (2013), Visterin (2014), Visterin (2015) and Stoffels (2016)
However, we believe the results are not completely correct, especially the numbers for 2012 and 2013 give a completely different view than the numbers for 2014-2016.

employees) which are the companies that use multiple GAAP reporting. Third, we are also able to interview an expert from SAP Belgium, which provides us with additional insights into the problem of multiple GAAP. Fourth, since our work is exploratory in nature, we believe that focusing on SAP can be justified because of its strong position in the market (and presence in our case studies).

4.4 The AIS Designs

Before we describe the AIS designs from the case studies, we first discuss some general observations, which we can draw based on the case studies:

- In general, companies try to limit differences between GAAP: when they have a choice, they choose a method that is allowed in all the GAAP they apply.
  - For example, using FIFO inventory measurement because it is allowed by both Belgian GAAP and IFRS, whereas LIFO is not allowed by IFRS.
  - For example, using the diminishing balance depreciation method, because it is allowed (and beneficiary) by tax legislation to limit the differences between Belgian GAAP and tax legislation.
- Differences between GAAP are mainly about capital grants, financial instruments, provisions, deferred taxes and pension benefits.

- Generally speaking, companies use two different GAAP. In the case studies, we observed the use of Belgian GAAP combined with IFRS or another European GAAP. Except for the insurance company: as from 2016 it has to comply to Solvency II, which can be considered an additional GAAP. Tax legislation is not considered as a separate GAAP, and is hence not included in the AIS, rather spreadsheets[1] are used to calculate needed items for tax reporting.
- In general, companies choose one GAAP as their primary GAAP, according to which they post all operational journal entries daily and report internally. Two of the cases choose Belgian GAAP, the other three IFRS as their primary GAAP. The choice for IFRS as primary GAAP is imposed by the respective parent companies of the case companies, the two other cases were not forced by their parent company to switch to IFRS.
- Out of five case companies two use a custom-built AIS, the other three use SAP.

To describe designs of AIS that report according to multiple GAAP, we use case studies and expert interviews. Additionally, we use the SAP website (SAP AG, 2013) to gain background information, since SAP is used by three cases and SAP has a market leader position in the segment of companies that use multiple GAAP (the use of SAP in this dissertation is further explained in Section 4.3).

By[2] analyzing the case material and consulting the SAP website (SAP AG, 2013), we identify the main components that all AIS from the cases have in common: ledger, accounts, charts of account, entry processing module, journal entry, company code and event. Together these components constitute the modular structure we study. We graphically represent this modular structure in Figure 4.1. Note that in this modular structure, we consider *"event"* as a given, since we do not consider how these events are recorded in the AIS: in the cases we focus on the processing of recorded events in multiple GAAP. This is logical since an AIS is not concerned with the recording of these events, this is done in more operational modules like a Sales module: the AIS takes into account these events and needs to process them in multiple GAAP for reporting purposes. Also, we already noted in the literature review that more research (mainly REA literature) has been conducted about how to record events so the needed accounting conclusions can be drawn, but research is lacking in how to actually make these accounting conclusions. Therefore, we repeat the definition of an event from REA literature: *"Economic Event represents either an increment or a decrement in the value of economic resources that are under the control of the enterprise."* (Hruby et al., 2006, p. 4).

An entry processing module contains five tasks (entry processing tasks) that need to be carried out to determine how to process events, according to the GAAP used. These tasks are: identification and qualification of concepts, recognition criteria, measurement methods, presentation requirements and disclosure requirements. The result of the entry processing tasks is a journal entry, consisting of the accounts that increase and/or decrease and the respective amounts. This implies that we do not consider presentation and

[1]The companies from our case study indicate that they use spreadsheets, because 1) they are easier to control than adding tax legislation into the AIS and 2) tax reports are only made up once a year, so the benefits of adding tax as a separate GAAP do not outweigh the effort/cost.

[2]The remainder of this paragraph and the following 3 paragraphs are (almost) a copy of Section 2.5.1.

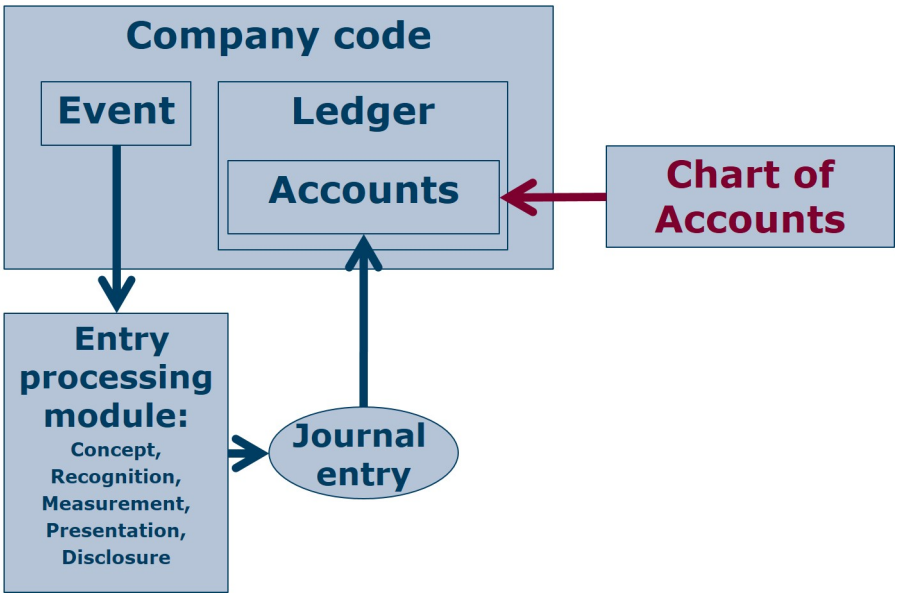

Figure 4.1: Modular structure

disclosure issues that result in other requirements (for example, additional information that needs to be provided in the notes) than posting a journal entry.

Journal entries are posted to a ledger. Next to containing all journal entries, the ledger keeps the totals of each account by adding all amounts posted to the account. The accounts used by the journal entries and collected in the ledger, are hierarchically structured according to a chart of accounts (with an account name and an account number) using a logical categorization of accounts (for example, assets, equity, liabilities, expenses and income).

Company codes are used to separate the financial information of multiple legally separated entities in the same system. Hence, events are always recorded and processed in a specific company code. Therefore, a ledger belongs to one company code. But one company code can hold multiple ledgers. A chart of accounts and an entry processing module can be used by ledgers from multiple company codes. In SAP (SAP AG, 2013) two additional principles have to be applied:

- Within the same company code, all ledgers need to use the same chart of accounts.
- In order to aggregate financial information from different company codes a separate consolidation module is required.

The modular structure described above is generic for reporting in one GAAP and needs to be expanded to support multiple GAAP. This can be done in multiple ways which requires making two design choices: the account design and the posting design. In Sections 4.4.1 and 4.4.2, we discuss these design choices and their implementation in the case studies. In Chapter 6, we use the modular structure to identify data elements, their relationships and attributes, to be able to design and construct the prototype. We graphically represent the modular structure in an **E**ntity **R**elationship **D**iagram (ERD) in Figure 6.1.

4.4.1 Account Designs

Different GAAP can require different processing of the same event. Therefore, a design choice needs to be made to post the journal entries of different GAAP to separate accounts, which we denominate the account design. In our case studies and from the SAP website (SAP AG, 2013), we identify three ways to separate accounts: 1) *parallel accounts* (account design 1): accounts for the different GAAP are duplicated in the same chart of accounts; 2) *parallel ledgers* (account design 2): journal entries for different GAAP are posted to separate ledgers and 3) *separate company codes* (account design 3): journal entries for different GAAP are posted to separate company codes. Each of these designs can be presented as an extension of the modular structure of Figure 4.1. We discuss these designs respectively in the following Sections 4.4.1.1, 4.4.1.2 and 4.4.1.3. When we illustrate the designs, we assume reporting is required in two GAAP.

4.4.1.1 Account Design 1 (a and b): Parallel Accounts

There are two ways to implement parallel accounts. The first possibility is used in two of the cases: they duplicate accounts (account design 1a), which means that for each GAAP all accounts are duplicated within the same chart of accounts.

If we relate this to the modular structure as presented in Figure 4.1, this design is related to the concept of *"Accounts"* in that modular structure: instead of having one set of accounts, an additional set of accounts is used in the same ledger. The actual extension of the modular structure is situated in the chart of accounts where accounts are duplicated: each GAAP has their own set of accounts in the same chart of accounts.

To distinguish the account numbers for the different GAAP, the first digit is used, the other digits of the account number resemble for corresponding accounts. We graphically represent an example of this account design in Figure 4.2. In this figure, we use 1, 2, 3 or 4 as the first digit for the accounts of the primary GAAP and 5, 6, 7 or 8 for the accounts of the secondary GAAP. For example, a revenue account *"Revenue product X"* exists twice in the chart of accounts with the number 4700000 for the primary GAAP and 8700000 for the secondary GAAP.

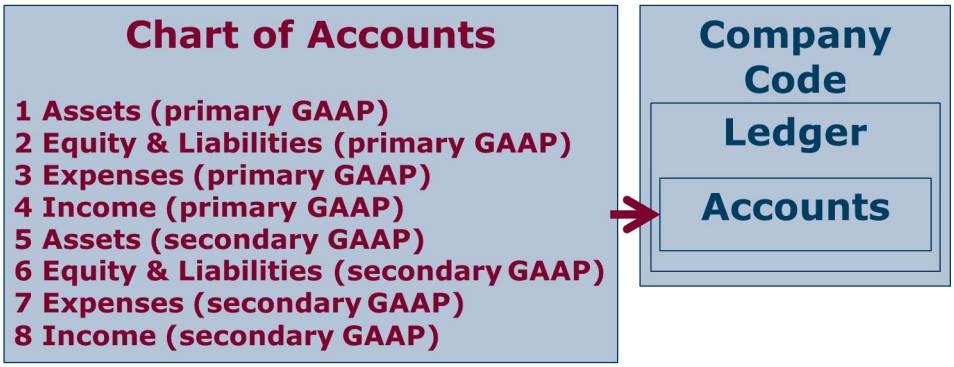

Figure 4.2: Account design 1a: duplicated parallel accounts

A second possibility to implement parallel accounts is the use of areas (account design 1b). Since none of the cases uses this design, we use the description on the SAP website (SAP AG, 2013): one chart of accounts is divided into areas, being a common area and a GAAP-specific area for each GAAP in which the company needs to report. The common area contains the accounts to which journal entries are posted that are the same for all GAAP. When the different GAAP require different journal entries, those entries are posted to the accounts in the GAAP-specific areas.

Just as account design 1b, the extension of the modular structure to set up multiple GAAP is situated in the chart of accounts in which the accounts used in the ledger are listed. The number of accounts that are added depends on the number of GAAP in the system: there are common accounts in the chart of accounts and specific accounts for each GAAP.

To distinguish the areas, for example, the first digit of the account number designates whether the account belongs to the common area or a GAAP-specific area, the other digits of the account number resemble for corresponding accounts. We graphically represent an example of this design in Figure 4.3: the first digit designates whether the account belongs to the common area (0), the primary GAAP area (1) or the secondary GAAP area (2). For example, when the journal entry for revenue posting is the same (same amount and timing) for all GAAP, the revenue account belongs to the common area and has the number 04700000. When the journal entry is different, two revenue accounts exist in the GAAP-specific areas with the numbers 14700000 and 24700000.

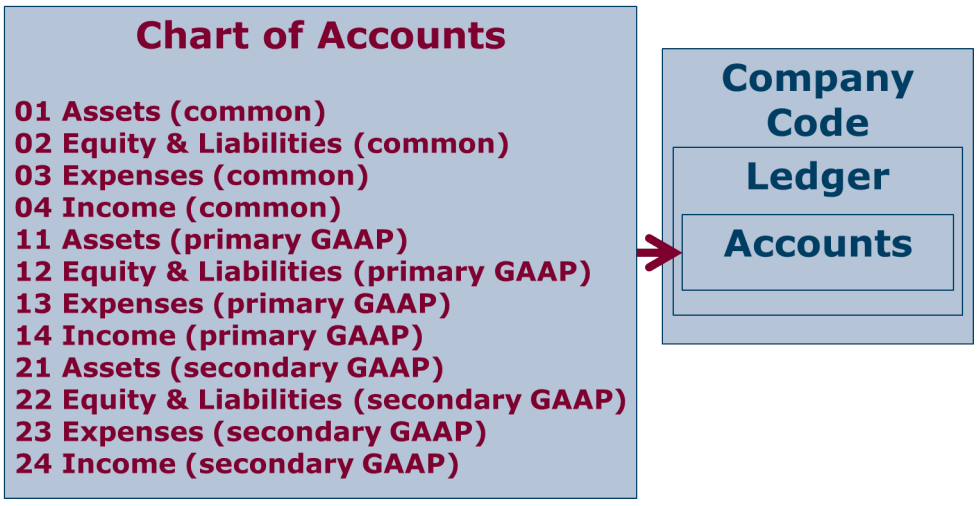

Figure 4.3: Account design 1b: parallel accounts, areas design

4.4.1.2 Account Design 2: Parallel Ledgers

The use of parallel ledgers (account design 2) is available since version 5.0 (released in 2004) of SAP. Because three cases already reported in multiple GAAP before they implemented version 5.0 of SAP, they use a different account design. Therefore, none of the cases use the parallel ledgers account design. Hence, we use the description on the SAP website (SAP AG, 2013) to describe the account design: every GAAP has a

separate ledger and all ledgers use the same chart of accounts. Every account occurs once in the chart of accounts and is used by both ledgers. If we relate this design to the modular structure of Figure 4.1, it is extended by adding ledgers so each GAAP has its own ledger. We graphically represent the parallel ledgers account design in Figure 4.4.

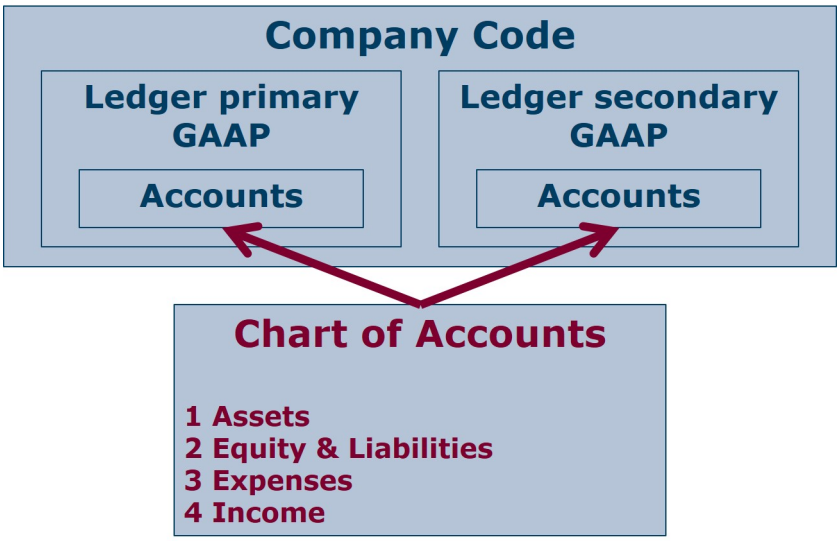

Figure 4.4: Account design 2: parallel ledgers

In certain cases, an account is only needed for a certain GAAP. For example the account *"Impairment of treasury shares"* is needed for a Belgian GAAP ledger, but cannot be used in an IFRS ledger, since IFRS does not allow to impair treasury shares. In this case the account specific for Belgian GAAP can be configured so it can only be used for the Belgian GAAP ledger and not for the IFRS ledger. Another example are the specific accounts needed for Other Comprehensive Income in IFRS, these kinds of accounts do not exist in other GAAP, like for example in Belgian GAAP.

4.4.1.3 Account Design 3 (a, b and c): Separate Company Codes

In versions of SAP before 5.0 and the custom-built software packages of the case companies it is not possible to create multiple ledgers in the same company code. That is why we find three account designs where separate company codes are used to separate accounts from different GAAP for the same company (legal entity).

Account design 3a is a design from SAP and is not used by any of the case studies. However, it is described on the SAP website (SAP AG, 2013) as a design in which a separate company code is used for each GAAP of the same company (legal entity) and where each company code uses a different chart of accounts. The modular structure (of Figure 4.1) is extended by adding company codes so each GAAP has its own company code. Moreover, additional charts of accounts are added so each company code uses its own chart of accounts. We graphically represent account design 3a in Figure 4.5. As mentioned before, to report financial information from different company codes in one report, SAP requires consolidation.

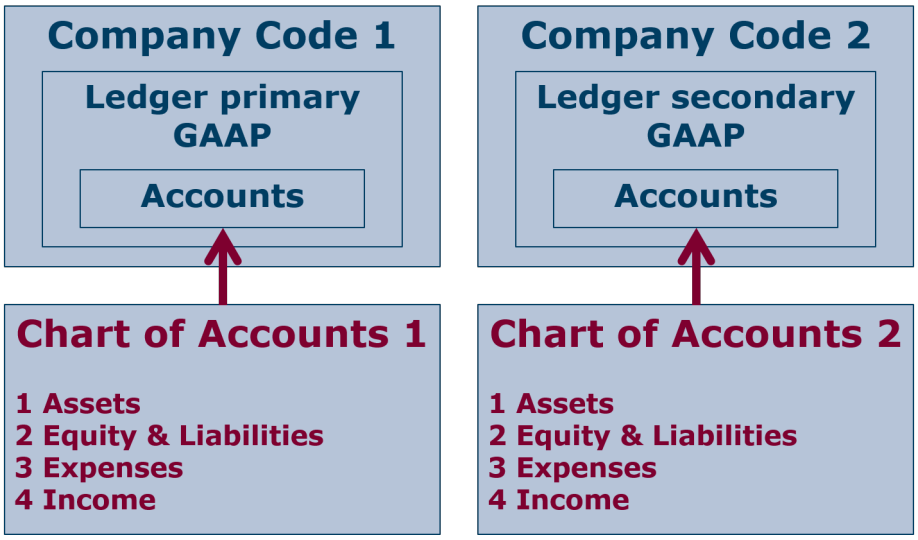

Figure 4.5: Account design 3a: separate company codes, different chart of accounts

In account design 3b multiple company codes are created for each GAAP, but the company codes use the same chart of accounts. This design is also possible in SAP (SAP AG, 2013), however, this design is used in the two case companies that have custom-built software. The custom-built software of the case studies does not require consolidation to report financial information from multiple company codes, like SAP does (SAP AG, 2013). The methods they use for reporting can be compared to the methods used by SAP to report financial information for parallel ledgers. Therefore, the design of the custom-built software from the cases strongly resembles account design 2, because the only difference is the way they call the separation of the ledgers: as a parallel ledgers design or as a company codes design.

The modular structure (of Figure 4.1) is extended by adding company codes so each GAAP has its own company code. Each of these company codes uses of the same chart of accounts. We graphically represent account design 3b in Figure 4.6.

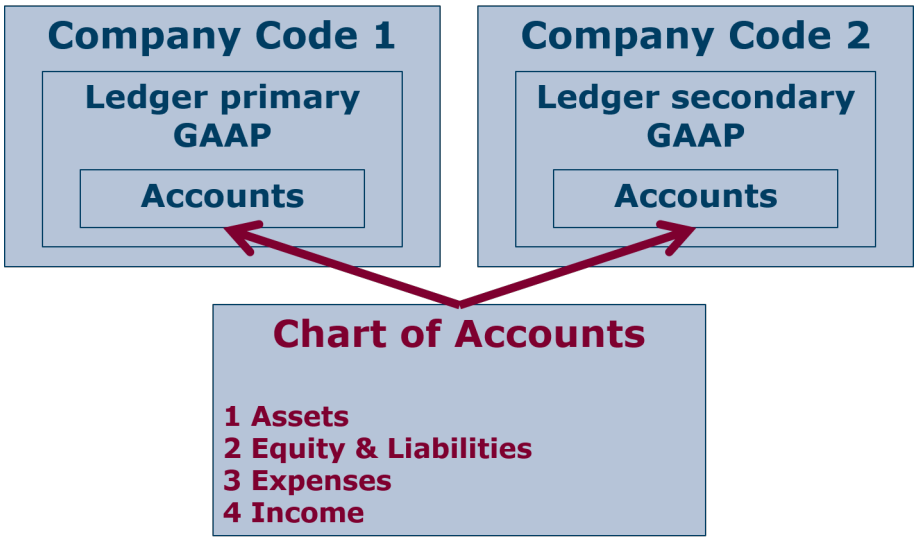

Figure 4.6: Account design 3b: separate company codes, same chart of accounts

Account design 3c is used by one of the case companies. The design requires four company codes: 1) operational transactions for the primary GAAP, 2) operational transactions for the secondary GAAP, 3) overhead related transactions for the primary GAAP and 4) overhead related transactions for the secondary GAAP. Respectively, these company codes are based on the same chart of accounts: one chart of accounts for overhead and one for operational transactions. In this account design the modular structure (of Figure 4.1) is extended by adding company codes and charts of accounts so each GAAP has a company code for both overhead and operational transactions linked to the respective chart of accounts. We graphically represent account design 3c in Figure 4.7.

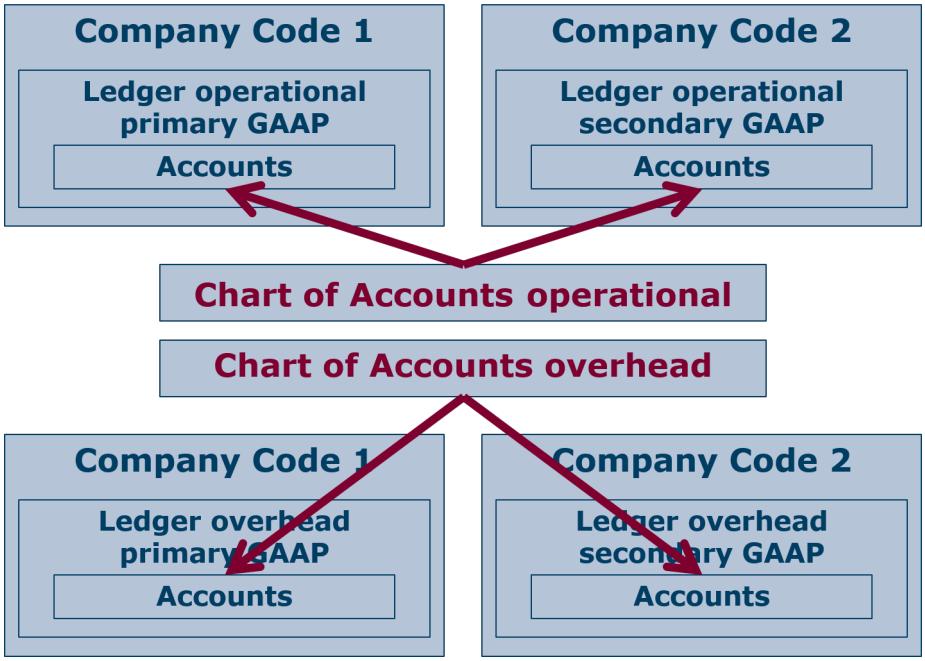

Figure 4.7: Account design 3c: separate company codes and separation operational and overhead

4.4.2 Posting Designs

The second design choice to set up multiple GAAP AIS, is the posting design. We discuss four different ways to post journal entries to a ledger, which we identified in our case studies and from the SAP website (SAP AG, 2013). In case of difference posting (posting design 1), the appropriate accounts need to be selected for reporting, how this is done is described in Section 4.4.2.2.

4.4.2.1 Posting Design 1: Difference Posting

Difference posting means that all journal entries are posted to the accounts of the primary GAAP and only the difference between the primary and secondary GAAP is posted to the accounts of the secondary GAAP.

For example: linear yearly depreciation of a machine (acquisition cost €100,000) is 20% in the primary and 25% in the secondary GAAP. If we use account design 1a from Figure 4.2 this results in the following journal entries:

To the primary ledger:

3630200	Depreciation of non-current fixed assets	20,000	
@ 1241209	Machines - accumulated depreciation		20,000

To the secondary ledger (only the difference is posted):

7630200	Depreciation of non-current fixed assets	5,000	
@ 5241209	Machines - accumulated depreciation		5,000

Note that, if the example would be: linear yearly depreciation of a machine (acquisition cost €100,000) is 25% in the primary and 20% in the secondary GAAP, the resulting journal entries are:

To the primary ledger:

3630200	Depreciation of non-current fixed assets	25,000	
@ 1241209	Machines - accumulated depreciation		25,000

To the secondary ledger (only the difference is posted):

5241209	Machines - accumulated depreciation	5,000	
@ 7630200	Depreciation of non-current fixed assets		5,000

To report according to a specific GAAP, the appropriate accounts need to be selected. Which accounts need to be selected depends on the account design and the selection method applied. We discuss the selection methods in the next section.

4.4.2.2 Selection methods

There are two possible approaches to select the appropriate accounts when reporting in a specific GAAP: selection method 1) determine which accounts to include and/or add and selection method 2) add all accounts, except for specific accounts not used in one of the GAAP. The first approach is applicable for account designs 1a and 1b, the second approach for account designs 2 and 3b.

We start out with the first approach (selection method 1): determine which accounts to include for account designs 1a and 1b. For the primary GAAP we only need to specify which accounts to select. For account design 1a, using Figure 4.2, we need to select all accounts starting with 1, 2, 3 and 4. For account design 1b, using Figure 4.3, we need to select the common accounts (01, 02, 03 and 04) and the GAAP-specific accounts (11, 12, 13 and 14).

For the secondary GAAP, we need to add accounts from the primary and the secondary ledger. There are two ways to do this, which we distinguish as selection method 1a and 1b. First, **selection method 1a**: selection by a detailed list with specification of accounts to add. Applied to account design 1a from Figure 4.2, for example: account 4700000 needs to be added with 8700000, account 4710000 with 8710000, etc. Applied to using account

design 1b from Figure 4.3, all common accounts (starting with 0) need to be selected and the accounts of the GAAP-specific areas need to be added pairwise, defined as a list: for example, account 11700000 needs to be added with 21700000, account 11710000 with 21710000, etc.

Second, **selection method 1b**: selection by specifying ranges of accounts. These ranges need to be added pairwise. Applied to account design 1a from Figure 4.2 we add accounts starting with the number 1 with those starting with number 5 pairwise, 2+6, 3+7 and 4+8. In that way account 4700000 needs to be added with 8700000, account 4710000 with 8710000, etc. But compared to selection method 1a, this is defined by the ranges and not for each individual account. For account design 1b from Figure 4.3 we add accounts starting with 11 with those starting with 21, 12+22, 13+23 and 14+24, but we also need to select the common accounts (01, 02, 03 and 04).

When using account design 2, for the primary GAAP all accounts of the primary ledger need to be selected for reporting. For the secondary GAAP, accounts of the primary and secondary GAAP need to be added pairwise. In both cases, we need a way to exclude accounts that are not used in a specific GAAP, like in the impairment of treasury shares example of Section 4.4.1.2. We can do that in two ways, which we label selection method 2a and 2b: in **selection method 2a** a *"list of exceptions"* is used for every GAAP to exclude accounts not used in a GAAP (one list for every GAAP) and in **selection method 2b** upon creation it is indicated whether an account is to be used in the different GAAP.

In account designs 3a en 3b, for selecting the accounts for the primary GAAP, all accounts in the primary GAAP company code need to be selected. Selection method 2a or 2b is applicable if some accounts need to be excluded. For the secondary GAAP, account design 3b requires to add accounts of the primary and secondary GAAP company codes. Since these company codes use the same chart of accounts this is quite straightforward and selection method 2a and 2b can be used to exclude accounts. Account design 3a requires consolidation for selecting accounts for the secondary GAAP. Since the charts of accounts are different for the primary and the secondary GAAP, selection methods 1a and 1b are applicable to be able to report for the secondary GAAP.

For account design 3c, consolidation of two company codes is required for reporting in the primary GAAP. Since the accounts are complementary, only selection methods 2a and 2b might be applicable. For reporting in the secondary GAAP, account design 3c requires to consolidate four company codes: two for complementary accounts for both GAAP (overhead accounts and operational accounts) and two need to be added pairwise (the ones of the respective GAAP), so all four (1a, 1b, 2a and 2b) selection methods are relevant.

4.4.2.3 Posting Design 2: Complete Posting

The use of complete posting (posting design 2) means that separate journal entries are posted to the accounts of the different GAAP, even when journal entries are identical.

Applied to the depreciation example from Section 4.4.2.1, this results in the following journal entries:

To the primary ledger:

3630200	Depreciation of non-current fixed assets	20,000	
@ 1241209	Machines - accumulated depreciation		20,000

To the secondary ledger:

7630200	Depreciation of non-current fixed assets	25,000	
@ 5241209	Machines - accumulated depreciation		25,000

To report according to a specific GAAP, the balance of an account can be found on the GAAP-specific account when using posting design 2. No adding off accounts is needed, therefore, the selection methods of Section 4.4.2.2 are not relevant. For example, for the account with the number 5241209, the amounts posted to this account simply need to be added to obtain the balance of the account for reporting.

4.4.2.4 Posting Design 3

Because of major differences between Belgian GAAP and IFRS regarding financial instruments, the insurance company from the case studies chooses to process every event related to financial instruments independent for both GAAP. Posting design 3 uses complete posting (posting design 2) for financial instruments, whereas all other journal entries are posted by difference (posting design 1). But this requires a way to handle the mixing of the two posting designs. Therefore, for all journal entries that use complete posting (in this case the ones related to financial instruments) an additional journal entry is posted to the secondary ledger, which is the reverse of the journal entry to the primary ledger. The two journal entries (primary GAAP and reverse to secondary GAAP) eliminate each other when the ledgers are added for reporting.

We illustrate posting design 3 by using the same depreciation example as for posting design 1 and 2 (see Section 4.4.2.1 and 4.4.2.3). Under the assumption that in this example posting design 3 is used for fixed assets. This results in the following journal entries:

To the primary ledger:

3630200	Depreciation of non-current fixed assets	20,000	
@ 1241209	Machines - accumulated depreciation		20,000

To the secondary ledger: the reverse of the journal entry to the primary ledger and the journal entry for the secondary GAAP:

5241209	Machines - accumulated depreciation	20,000	
@ 7630200	Depreciation of non-current fixed		20,000

7630200	Depreciation of non-current fixed assets	25,000	
@ 5241209	Machines - accumulated depreciation		25,000

4.4.2.5 Posting Design 4

One of the case studies uses difference posting, except for postings regarding fixed assets. This case company uses the *"asset accounting"* module of SAP for managing and controlling their fixed assets and to record all details of related transactions. Therefore, they use complete posting (posting design 2) for fixed assets. Journal entries for the fixed assets use posting design 2 (also see examples from Section 4.4.2.3), all other journal entries use posting design 1 (also see examples from Section 4.4.2.1).

In posting design 4 the mixture of two posting methods is handled by adjusting the selection criteria for reporting: for the primary GAAP all primary GAAP accounts are selected, for the secondary GAAP accounts of the primary and secondary GAAP need to be added pairwise (like for posting design 1), except for the accounts for which posting design 2 is used (in the example the fixed assets): there only the accounts for the secondary GAAP need to be selected.

4.5 Evaluating the Case Study Results, using the Framework for Evaluation in Design Science Research (FEDS)

In this section, we discuss how we evaluate the designs from the case studies of the previous section. After each interview with the case companies, we analyze the notes to identify the structure of the AIS design of the case (Section 4.4). Then we evaluate these designs with regard to evolvability. Since the AIS designs can be considered artifacts, we use the FEDS of Venable et al. (2014) to increase the validity of the analysis. In this framework, four steps need to be executed to evaluate an artifact.

The first step is to explicate the goals of the evaluation. Our goals are to reveal combinatorial effects in existing AIS and to provide evidence for the relevance of Normalized Systems Theory in the accounting domain.

Step two is to choose an evaluation strategy consisting of why, when and how to evaluate. Since we want to determine whether the case designs can be used as a starting point for further development of evolvable multiple GAAP AIS, we use a summative (why) ex-ante (when) evaluation. We evaluate (how) by proposing (theoretical) changes to the designs of the AIS and ascertain whether the changes cause combinatorial effects in the respective designs. We report on this evaluation in Section 4.6. This can be classified as an artificial evaluation, since we do not test the AIS (and the proposed changes) in a natural setting.

Step three is to determine the properties to evaluate. The first goal is the most important one: the property we want to evaluate is whether the AIS from the case studies contain combinatorial effects.

Fourth and last step is to design the individual evaluation episodes, these are reported in Section 4.6. In these evaluation episodes we first conduct a within-case analysis of

each design. But we use a flexible and opportunistic data analysis (Benbasat et al., 1987; Dubé and Paré, 2003; Eisenhardt, 1989; Yin, 2003): when revealing a combinatorial effect in a case, we evaluate whether it also exists in previously analyzed cases. Then we structure findings by cross-case comparison and by conducting additional analysis. This results in the description of different structures used to set up multiple GAAP AIS and the combinatorial effects resulting from different changes imposed on these structures. Lastly, we use insights from the expert interviews and online documentation of SAP to review and extend the analysis.

The use of Normalized Systems Theory increases the rigor (Hevner, 2007) of this work: we use an existing design theory (classified as prescriptive knowledge by Gregor and Hevner (2013)). Since we select a practical problem (multiple GAAP, described in more detail in Chapter 1), we can argue that this work is relevant (Hevner, 2007). Moreover, the combinatorial effects we identify contribute additionally to the relevance of this work, since they show that evolvability is an issue in current multiple GAAP AIS. In the classification of Gregor and Hevner (2013), the combinatorial effects are a contribution to the descriptive knowledge base. Situating and relating different implementations of AIS in practice, can also be classified as a level 1 contribution to the prescriptive knowledge base (Gregor and Hevner, 2013).

4.6 The Combinatorial Effects

In this chapter we discuss the combinatorial effects that occur when we apply changes to the designs we discussed in Section 4.4. We evaluate the impact of the following changes:

1. Creating a new account;
2. New version of an entry processing task for one GAAP (effect on journal entries);
3. New version of an entry processing task for one GAAP (effect on entry processing module);
4. New version of an entry processing task for all GAAP;
5. Adding a new GAAP to the AIS.

To explain why we use these changes, we refer to the problem domain we address: *"The design of evolvable AIS that support processing of accounting-relevant events in multiple GAAP"*. The focus of this dissertation is on the processing of accounting-relevant events. Therefore, we consider changes caused by GAAP that affect the processing of the accounting-relevant events. These GAAP changes can be changes in the GAAP themselves or a change of the GAAP in which the company needs to report. A change in a GAAP can always be expressed as a change of one or more entry processing tasks (concept, recognition, measurement, presentation and disclosure) and it might require adding a new account to the system. This explains the origin of the first four changes. Creating a new account can also have another trigger: for example, a company starts selling a new product and wants to register revenue for that product on a separate account, then a new account needs to be added to the AIS. The fifth change reflects a change in the GAAP a company needs to report in. We consider these changes as the anticipated changes

for a multiple GAAP AIS: this means that the system should be able to handle these changes without causing combinatorial effects. If combinatorial effects do occur when these changes are imposed on the system, the system is not considered evolvable.

For every change we discuss the impact of the change in different designs by illustrating different contexts. We assume the use of two GAAP: the primary and the secondary GAAP. We identify a combinatorial effect if the impact of the change is not only proportional to the complexity of the change itself but is also proportional to the size of the system on which it is imposed. One possible variable that is related to the size of the system is the number of used GAAP: this means a combinatorial effect is caused when the impact of the change is proportional to the number of GAAP in the system.

4.6.1 Change 1: Creating a New Account

The first change we consider, is the creation of a new account. For example, a company sells two products for which separate accounts exist in the AIS. They start selling a third product and want to post revenue for that product on a separate account.

4.6.1.1 Context 1: Impact on Duplicated Parallel Account Design (Account Design 1a)

To create a new account in account design 1a, a new account needs to be created in every duplicated set of accounts. In the modular structure (as shown in Figure 4.1 and extended in Figure 4.2), the new account needs to be added for each GAAP in the same chart of accounts. For example, revenue for the two existing products is posted to the accounts with the numbers 4700000 and 4701000 for the primary GAAP and to the accounts with the numbers 8700000 and 8701000 for the secondary GAAP. For the new product, account 4702000 is created for the primary GAAP and 8702000 for the secondary GAAP. Since the impact of creating a new account (here two accounts need to be created) in account design 1a is proportional to the number of used GAAP (two in our example), the change causes a combinatorial effect.

4.6.1.2 Context 2: Impact on Parallel Account Design using Areas (Account Design 1b)

The impact of creating a new account using account design 1b, depends on where the new (revenue) account needs to be added: in the common area or in a GAAP-specific area.

In the common area: (revenue) accounts belong to the common area when journal entries (for revenue) are the same for different GAAP. In the modular structure (as shown in Figure 4.1 and extended in Figure 4.3), the new account needs to be added in the common area of the chart of accounts. For example, revenue accounts have the numbers 04700000 and 04701000. Adding a new revenue account requires creating a

new account in the common area with the number 04702000. No combinatorial effect is induced, since the impact of the change is only proportional to the change itself (creating one new account).

In a GAAP-specific area: separate (revenue) accounts exist if GAAP require a different processing (of revenue). In the modular structure, the new account needs to be added in each GAAP-specific area of the chart of accounts. For example, because in one GAAP revenue is recognized when the invoice is drafted and in another GAAP when the goods are delivered. Using Figure 4.3, the revenue accounts have the numbers 14700000 and 14701000 for the primary GAAP and 24700000 and 24701000 for the secondary GAAP. For the new product, two new accounts need to be created: 14702000 and 24702000. This is a combinatorial effect, because new accounts need to be created for each GAAP: the impact is proportional to the number of GAAP used.

4.6.1.3 Context 3: Impact on Selection Method 1

The change also has an impact on the selection criteria for reporting. In case account design 1a or 1b is used, only selection method 1 is relevant. When selection method 1a is used, the newly created account(s) need(s) to be included into the reports of the respective GAAP: the account(s) need(s) to be added to the list of *"accounts to include"* for each GAAP that uses that account. A combinatorial effect occurs, since the impact is proportional to the number of used GAAP: every list of each GAAP needs to be updated. When selection method 1b is used, the created accounts fall within the defined ranges as described in Section 4.4.2.1, hence the change does not result in a combinatorial effect.

4.6.1.4 Context 4: Impact on Parallel Ledgers (Account Design 2)

In a parallel ledger design, a new account needs to be added to the common chart of accounts in the modular structure (shown in Figure 4.1 and extended in Figure 4.4). For example, account numbers 4700000 and 4701000 are used for revenue of existing products and account number 4702000 is created for the new product. Both ledgers use these accounts. Since the impact of the change is only proportional to the change itself (creating one new account) and not to the system size, no combinatorial effect is caused.

4.6.1.5 Context 5: Impact on Selection Method 2

In case account design 2 is used, selection method 2 becomes relevant and might cause additional combinatorial effects. If the newly created account is an account that is only used by one GAAP and selection method 2a is used, the account needs to be added to the list of exceptions for all other GAAP. Since the impact of this change is proportional to the number of GAAP in the system, the change causes a combinatorial effect. When selection method 2b is used and upon creation of the account you need to indicate for which GAAP the account is used, the impact of the change is not proportional to the number of GAAP and hence does not cause a combinatorial effect.

4.6.1.6 Context 6: Impact on Separate Company Codes Design (Account Design 3a): Different Charts of Accounts

In account design 3a, the company codes all use a different chart of accounts. Therefore, the impact of creating a new account is proportional to the number of charts of accounts in use. Applied to the example and the modular structure of Figure 4.1, extended by Figure 4.5: in chart of accounts 1 and 2, two accounts exist for the revenue of the existing products with the account numbers 4700000 and 4701000. For the new product, both charts of accounts need to add a new revenue account with the number 4702000. Consequently, a combinatorial effect is caused, since the impact of creating a new account is proportional to the number of charts of accounts (hence, the number of GAAP) used.

Regarding selection criteria, in account design 3a consolidation is required to add the different ledgers from the different company codes. Depending on the selection method(s) used in the consolidation description combinatorial effects are induced or not. We already discussed the combinatorial effects caused by selection methods 1a and 2a in Sections 4.6.1.3 and 4.6.1.5.

4.6.1.7 Context 7: Impact on Separate Company Codes Design (Account Design 3b): Same Chart of Accounts

When the same chart of accounts is used (account design 3b), the impact of creating a new account is limited to adding only one account. In the modular structure of Figure 4.1 and the extension of Figure 4.6, the account is added to the chart of accounts, which is the only chart used by the different company codes. For example, account numbers 4700000 and 4701000 are used for revenue of existing products and account number 4702000 is created for the new product. Both company codes use these accounts. Since the impact of the change is limited to creating one new account, no combinatorial effect is induced by the change. The selection method that is applied, might induce a combinatorial effect, as described in Section 4.6.1.5.

4.6.1.8 Context 8: Impact on Separate Company Codes Design (Account Design 3c): Separate Overhead and Operational Chart of Accounts

In account design 3c the company codes use the same chart of accounts, but there is a separate chart of accounts for operations and for overhead. Since an account is exclusively present in the operations chart or the overhead chart, the impact of adding a new account is the same as in account design 3b. In the modular structure (as shown in Figure 4.1 and extended in Figure 4.7), the new account is added to or the overhead chart of accounts or the operational chart of accounts. For example, account numbers 4700000 and 4701000 are used for revenue of existing products in the operational chart of accounts and account number 4702000 is created in the operational chart of accounts for the new product. Since the impact of the change is limited to creating one new account, no combinatorial effect is induced by the change.

Regarding selection criteria, in account design 3c consolidation is required to add the different ledgers from the different company codes. Depending on the selection method(s) used in the consolidation description combinatorial effects are induced or not. We already discussed the combinatorial effects caused by selection methods 1a and 2a in Sections 4.6.1.3 and 4.6.1.5.

4.6.1.9 Summary of Impact of First Change (Creating a New Account)

In Table 4.2, we provide an overview of the impact of creating a new account when using the different account designs and the additional impact, depending on the selection method.

Context	**Impact of creating a new account**	
	CE	**No CE**
Account design 1a	x	
Account design 1b: add account in ⇒ common area		x
⇒ GAAP-specific area	x	
Account design 2		x
Account design 3a	x	
Account design 3b		x
Account design 3c		x
Selection method 1a	x	
Selection method 1b		x
Selection method 2a	x	
Selection method 2b		x
Consolidation	Depends on which selection method is used for consolidation; impact of all selection methods (1a, 1b, 2a and 2b) can be relevant	

Table 4.2: Summary table of the impact of change 1: creating a new account

4.6.2 Change 2: New Version of an Entry Processing Task for One GAAP (Effect on Journal Entries)

In this section we consider that a new version of an entry processing task for one GAAP is introduced, but we only consider the effect on journal entries. We illustrate this change with changing revenue recognition criteria, using the following example: a company sells goods for €20,000: the sales contract is drafted on 01/08/2016, the goods are delivered on 03/08/2016 and the invoice is drafted on 15/08/2016. We assume, like in change 1, that revenue for the two existing products is recorded in separate accounts and reporting is required according to two GAAP.

When we discuss this change, we need to take into account four possible situations. Firstly, the impact is different whether the entry processing task changes for the primary

or secondary GAAP. Secondly, there is a difference whether the criteria were the same in the past for both GAAP or the journal entry between the GAAP was already different. We label these four possible situations A, B, C and D and represent them in Table 4.3.

		Of which GAAP do the recognition criteria change?	
		Primary	Secondary
Journal Entry in the past was:	The same	A	B
	Different	C	D

Table 4.3: Possible situations when recognition criteria change

We apply the situations (Table 4.3) to the example before the change: for situations A and B revenue is recognized when the invoice is drafted and for situations C and D revenue is recognized for the primary GAAP when the sales contract is signed and for the secondary GAAP when the invoice is drafted. In all four situations we consider the change of revenue recognition criteria for a GAAP from their original criteria to recognition when the goods are delivered[3].

4.6.2.1 Context 1: Impact on Account Design

We evaluate the impact of a changing entry processing task on the account designs. For account design 1b and in situations A and B (Table 4.3) the initial journal entries are the same and the revenue accounts belong to the common area. For example, using Figure 4.3, the revenue accounts have the numbers 04700000 and 04701000. When revenue recognition criteria change, new accounts need to be created in all GAAP-specific areas, since the ones from the common area can no longer be used. For example, 14700000, 14701000, 24700000 and 24701000 are created. Because the impact is proportional to the number of GAAP, a combinatorial effect arises. In situations C and D (Table 4.3) the revenue accounts already belong in the GAAP-specific areas. Hence, no additional accounts need to be created.

For account designs 1a, 2, 3b and 3c, it is not necessary to create new accounts, since the needed accounts already exist: for account design 1a accounts are duplicated and account designs 2, 3b and 3c use the same chart of accounts.

Lastly for account design 3a, the impact depends on the way accounts are duplicated. Firstly, when accounts are duplicated in all charts of accounts, no new accounts need to be created when a new version of the revenue recognition criteria is imposed. Secondly, when only the accounts needed for difference posting (posting design 1) are duplicated, the impact depends on the situation as depicted in Table 4.3. In situation A, a new account needs to be added for each additional GAAP. For example, in chart of accounts 1, revenue is posted to accounts 4700000 and 4701000. After the change, the accounts 4700000 and

[3]The criteria for revenue recognition usually require a *"transfer of the significant risks and rewards of ownership of the goods"* (IASB, 2001) like in the IAS 18 Revenue standard, which is amended in IFRS 15 to the notion of a performance obligation that refers to a transfer of control over the goods (IASB, 2014). However, other standards might not include specific guidance on the issue. For example, Belgian accounting legislation does not address the issue of revenue recognition.

4701000 need to be created in chart of accounts 2. This results in a combinatorial effect since the impact is proportional to the number of charts of accounts (and hence, the number of used GAAP). In situation B, the impact is limited to creating accounts for the GAAP of which the revenue recognition criteria changes. In situations C and D, separate accounts already exist, since there was already a difference in journal entries in the past (in the example for revenue). So no combinatorial effect arises in situations B, C and D.

In the following contexts we use account design 2 and related Figure 4.4 to illustrate the impact of the new version of an entry processing task, to avoid combinatorial effects related to the other account designs.

4.6.2.2 Context 2: Impact on Difference Posting (Posting Design 1)

Here we discuss the impact of a new version of an entry processing task on difference posting. The impact does not depend on the account design, but does depend on the situations from Table 4.3. We illustrate this context, using the same example as in Section 4.6.2. In that example, revenue recognition criteria change to recognition at delivery date. Note that in this context, difference posting is used and hence only the difference between the GAAP is posted as a journal entry to the secondary ledger.

Before the change: journal entries for situations A and B, revenue recognition on invoice date

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

Date	**To the secondary ledger:**
01/08/2016	No entry
03/08/2016	No entry
15/08/2016	No entry

After the change: new journal entries for situation A, since revenue recognition criteria for the primary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000
15/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000

In this changed situation, revenue is recognized at delivery date for the primary GAAP. The criteria for the secondary GAAP did not change, but since difference posting is used (posting design 1), journal entries to the secondary ledger need to be added so for the secondary GAAP revenue is recognized at invoice date. We conclude that by changing the recognition criteria for the primary GAAP, the journal entries to both the primary and the secondary ledger need to be adjusted. This is because difference posting is used: the journal entries to the secondary ledger depend on the ones to the primary ledger. Changing the revenue recognition criteria for the primary GAAP changes the journal entries to the primary ledger and hence the ones to the secondary ledger need to be adjusted as well to make sure that revenue is recognized when the invoice is drafted upon reporting for the secondary GAAP. In the first journal entry to the secondary ledger, the journal entry to the primary ledger is reversed (revenue cannot yet be recognized for the secondary GAAP). In the second journal entry revenue needs to be recognized, taking into account the journal entries to the primary ledger.

We can expand this conclusion as follows: the impact of the change increases when the number of GAAP used increases (if all GAAP post the difference with regard to the primary GAAP). Since the impact of this change is not only proportional to the change itself, but also to the number of GAAP used, the change causes a combinatorial effect.

After the change: new journal entries for situation B, since revenue recognition criteria for the secondary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000

In situation B, the entries to the primary GAAP remain the same as before the change. Only the journal entries to the secondary GAAP change: an additional journal entry is posted on 03/08/2016 to recognize revenue when the goods are delivered. At 15/08/2016, the journal entry needs to be adjusted to avoid double reporting of revenue by the use of difference posting (primary and secondary ledger are added).

The impact of the changing recognition criteria is limited to the journal entries to the secondary GAAP ledger. Since the impact is not proportional to the number of GAAP, no combinatorial effect arises.

Before the change: journal entries for situations C and D, primary GAAP recognizes revenue when sales contract is signed, the secondary GAAP when the invoice is drafted

Date	**To the primary ledger:**			
01/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000
03/08/2016	No entry			
15/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000

Since only the difference between the primary and the secondary GAAP is posted to the secondary ledger, the first journal entry to the secondary ledger is the reverse of the entry to the primary ledger because revenue cannot yet be recognized. Revenue is only recognized when the invoices are prepared for the secondary GAAP.

After the change: new journal entries for situation C, since revenue recognition criteria for the primary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000
15/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000

We conclude that changing recognition criteria for the primary GAAP requires to adjust both the journal entries to the primary as to the secondary ledger. This is the result of

the use of difference posting: the journal entries of the secondary GAAP depend on those to the primary GAAP. The journal entries to the secondary ledger need to be adjusted in order to keep recognizing revenue when the invoice is drafted in the secondary GAAP. Since the impact of the change is not only proportional to the change itself, but is also proportional to the number of GAAP used (all GAAP that use difference posting with respect to the primary GAAP), the change causes a combinatorial effect.

After the change: new journal entries for situation D, since revenue recognition criteria for the secondary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	No entry			

In situation D, the entries to the primary GAAP remain the same as before the revenue recognition criteria changed. Only the journal entries to the secondary GAAP change: on 01/08/2016 the journal entry to the primary GAAP needs to be reversed, since revenue cannot yet be recognized, on 03/08/2016 revenue can be recognized, so this journal entry needs to be adjusted and on 15/08/2016 no journal entry is needed anymore. The impact of the changing recognition criteria is limited to the journal entries to the secondary GAAP ledger. Since the impact is not proportional to the number of GAAP, no combinatorial effect is induced.

Conclusion: impact of change on difference posting

In situations A and C, changing the revenue recognition criteria of the primary GAAP causes an impact on the journal entries to the primary GAAP. Since the postings of the secondary GAAP depend on those to the primary GAAP (use of posting design 1), they need to be changed as well. Therefore, the impact of changing revenue recognition criteria is proportional to the number of GAAP used (assuming all GAAP use difference posting with regard to the primary GAAP) for situations A and C, hence causing a combinatorial effect. In situation B and D, the change only requires adjusting the journal entries to the secondary ledger. Since the impact is limited to the journal entries of the changing GAAP, for situations B and D the change does not impose a combinatorial effect.

4.6.2.3 Context 3: Impact on Complete Posting (Posting Design 2)

Here we discuss the impact of a new version of an entry processing task on complete posting (posting design 2). We illustrate this context, using the same example as in Section 4.6.2. In that example, revenue recognition criteria change to recognition at delivery date. In the following paragraphs we provide all journal entries, using the different situations from Table 4.3.

Before the change: journal entries for situations A and B, revenue recognition on invoice date

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

After the change: new journal entries for situation A, since revenue recognition criteria for the primary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

The journal entries to the secondary ledger do not change, because they are posted independently. Since only the journal entries to the ledger of which the GAAP has changed, need to change, the impact of the change is only proportional to the change. No combinatorial effect arises, since the impact of the change is not proportional to the number of GAAP in use.

After the change: new journal entries for situation B, since revenue recognition criteria for the secondary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

The impact of the changing revenue recognition criteria is limited to the journal entries of the secondary GAAP. Since the impact is not proportional to the number of GAAP used, no combinatorial effect is induced.

Before the change: journal entries for situations C and D, primary GAAP recognizes revenue when sales contract is signed, the secondary GAAP when the invoice is drafted

Date	**To the primary ledger:**			
01/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

After the change: new journal entries for situation C, since revenue recognition criteria for the primary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

The impact of the changing recognition criteria is limited to changing the journal entries to the primary ledger. Since the impact is not proportional to the number of GAAP used, the change does not result in a combinatorial effect.

After the change: new journal entries for situation D, since revenue recognition criteria for the secondary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

The impact of the changing recognition criteria is limited to changing the journal entries to the secondary ledger. Since the impact is not proportional to the number of GAAP used, no combinatorial effect is induced.

Conclusion: impact of change on complete posting

Changing an entry processing task, in the example the revenue recognition criteria, does not result in a combinatorial effect in any of the situations (see Table 4.3) when complete posting (posting design 2) is used. This is because the journal entries to the different ledgers are posted independently. So, when the revenue recognition criteria change for one GAAP, only the journal entries for the respective GAAP need to be adjusted. The journal entries of the other GAAP remain unchanged, since they are also complete. No combinatorial effect arises, since the impact of the change is not proportional to the number of GAAP in use.

There are also no combinatorial effects related to selection methods, since these are not relevant in the context of complete postings: to report in a certain GAAP, all accounts from that GAAP need to be selected, no accounts need to be added or selected.

4.6.2.4 Context 4: Impact on Posting Design 3

In this section we discuss the impact of a new version of an entry processing task on posting design 3. We use the same example as in Section 4.6.2.2 and apply the change to all situations from Table 4.3. The use of posting design 3 is specifically for revenue posting, not for other postings. This means that for revenue three postings are made: one to the primary ledger, one to the secondary ledger and the reverse posting of the posting to the primary ledger to the secondary ledger.

Before the change: journal entries for situations A and B, revenue recognition on invoice date

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000
Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	4700000	Revenue product X	20,000	
	@ 1400000	Trade receivables		20,000
	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

After the change: new journal entries for situation A, since revenue recognition criteria for the primary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000
Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000
15/08/2016	1404100	Invoices to be prepared	20,000	
	@ 1400000	Trade receivables		20,000
	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

The change has an impact on the journal entries to the primary ledger and the reverse posting to the secondary ledger. When the reverse posting is generated automatically with the journal entry to the primary ledger, no combinatorial effect is caused. A combinatorial

effect would arise when the implementation would demand that the reverse posting needs to be adjusted for each GAAP separately: then the impact of the change is proportional to the number of GAAP used (and hence dependent on the size of the system).

After the change: new journal entries for situation B, since revenue recognition criteria for the secondary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

The journal entry to the primary ledger and the reverse entry to the secondary ledger remain unchanged. The change only causes an impact on the other journal entry to the secondary ledger: this impact is only proportional to the change and not to the number of GAAP used in the system, no combinatorial effect is induced.

Before the change: journal entries for situations C and D, primary GAAP recognizes revenue when sales contract is signed, the secondary GAAP when the invoice is drafted

Date	**To the primary ledger:**			
01/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000
03/08/2016	No entry			
15/08/2016	1404100	Invoices to be prepared	20,000	
	@ 1400000	Trade receivables		20,000
	1400000	Trade receivables	20,000	
	@ 4700000	Revenue product X		20,000

After the change: new journal entries for situation C, since revenue recognition criteria for the primary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	No entry			
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	No entry			
03/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000
	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1404100	Invoices to be prepared	20,000	
	@ 1400000	Trade receivables		20,000
	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

The impact of the change affects the journal entries to the primary ledger and the reverse posting to the secondary ledger. When the reverse posting is generated automatically, together with the journal entries to the primary ledger, no combinatorial effect arises. A combinatorial effect is caused when the implementation would demand that the reverse postings need to be adjusted for each GAAP individually: then the impact of the change would be proportional to the number of GAAP used (the size of the system).

After the change: new journal entries for situation D, since revenue recognition criteria for the secondary GAAP have changed

Date	**To the primary ledger:**			
01/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
03/08/2016	No entry			
15/08/2016	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

Date	**To the secondary ledger:**			
01/08/2016	4700000	Revenue product X	20,000	
	@ 1404100	Invoices to be prepared		20,000
03/08/2016	1404100	Invoices to be prepared	20,000	
	@ 4700000	Revenue product X		20,000
15/08/2016	1404100	Invoices to be prepared	20,000	
	@ 1400000	Trade receivables		20,000
	1400000	Trade receivables	20,000	
	@ 1404100	Invoices to be prepared		20,000

The journal entries to the primary ledger and the reverse postings to the secondary ledger remain unchanged. The change only causes an impact on the journal entries to the secondary ledger. Since this impact is only proportional to the change and not on the size of the system (number of GAAP used), no combinatorial effect is caused.

Conclusion: impact of change on posting design 3

In situations A and C, the changing revenue recognition criteria of the primary GAAP require adjusting the journal entries to the primary GAAP and the reverse posting to the secondary GAAP. If the reverse posting is generated automatically with the journal entry to the primary ledger, no combinatorial effect arises.

A combinatorial effect would be induced when the reverse posting needs to be adjusted for each GAAP separately: the impact of the change then is proportional to the number of GAAP used.

In situations B and D, the changing revenue recognition criteria only require changing the journal entries to the secondary ledger (the specific ones, not the reverse of the primary ledger entries). Since the impact is not proportional to the number of GAAP used, no combinatorial effect is caused.

4.6.2.5 Context 5: Impact on Posting Design 4

When posting design 4 is used, the impact of a new version of an entry processing task depends on the posting design used for the related event. If the related event uses difference posting (posting design 1), the impact of the change is the same as the impact for difference posting, which is discussed in Section 4.6.2.2. If the related event uses complete posting (posting design 2), the impact of the change is the same as the impact for complete posting, which is discussed in Section 4.6.2.3.

For example, changing revenue recognition criteria have a different impact depending on the posting design used for revenue related journal entries. If revenue uses difference posting (posting design 1), the impact is the same as the impact for difference posting. When complete posting (posting design 2) is used for revenue, the impact is the same as the impact for complete posting.

4.6.2.6 Summary of Impact of Second Change (a New Version of an Entry Processing Task for One GAAP - Effect on Journal Entries)

In Table 4.4, we provide an overview of the impact of a new version of an entry processing task for one GAAP when using the different account designs and posting designs in the different situations (see Table 4.3).

Context	**A**	**B**	**C**	**D**
Account design 1a				
Account design 1b	x	x		
Account design 2				
Account design 3a				
⇒ All accounts are duplicated				
⇒ Only accounts needed for difference posting are duplicated	x			
Account design 3b				
Account design 3c				
Posting design 1	x		x	
Posting design 2				
Posting design 3				
⇒ Reverse posting generated automatically				
⇒ Reverse posting not generated automatically	x		x	
Posting design 4	impact depends on posting design (posting design 1 or 2) used for the GAAP of which the entry processing task changes			

Table 4.4: Summary table of the impact of change 2: a new version of an entry processing task

Legend of symbols
x represents a combinatorial effect,
an empty cell represents that no combinatorial effect is caused by the change.

Legend of situations: also see Table 4.3
A: Primary GAAP changes, journal entry was the same in the past.
B: Secondary GAAP changes, journal entry was the same in the past.
C: Primary GAAP changes, journal entry was different in the past.
D: Secondary GAAP changes, journal entry was different in the past.

4.6.3 Change 3: New Version of an Entry Processing Task for One GAAP (Effect on Entry Processing Module)

As a third change, we consider the effect of a new version of an entry processing task for one GAAP on the entry processing module. The impact of this change is the same for all account and posting designs, but depends on the design of the entry processing module. Therefore, the contexts depend on whether the entry processing tasks are separated in distinct tasks (as shown in Figure 4.8) or not (all tasks belong to the same entry processing module, as graphically represented in Figure 4.1 of the modular structure). We use the examples from Section 4.6.2 to illustrate the contexts.

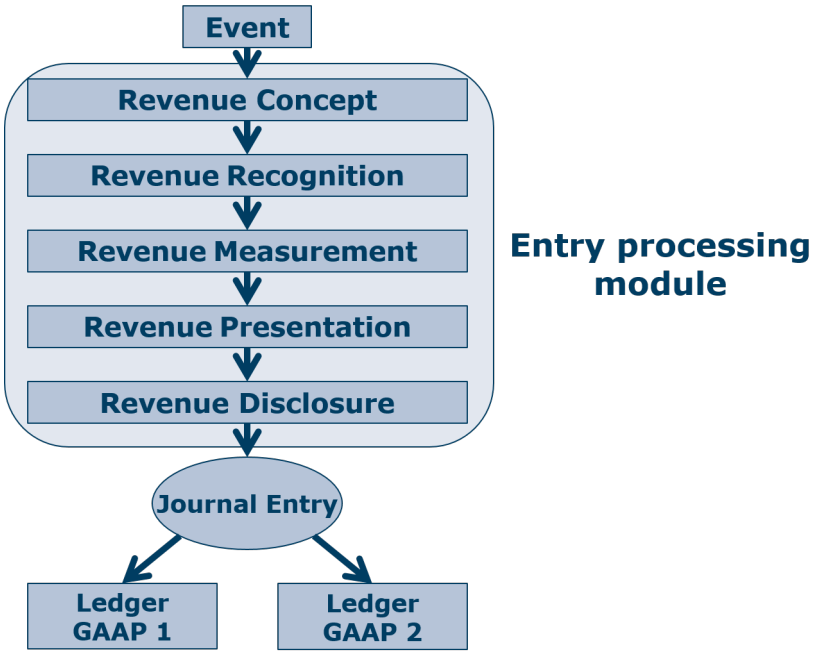

Figure 4.8: Separation of entry processing tasks

4.6.3.1 Context 1: Entry Processing Tasks not Separated

In this context, we assume that each concept has a specific entry processing module (also see Figure 4.1), with accompanying entry processing tasks. We assume a starting situation where the concept (for example, revenue) is processed in the same way for all GAAP, and therefore, there is only one entry processing module, used by all GAAP. When the new version of the entry processing task is introduced for the secondary GAAP, a new entry processing module needs to be created with the same tasks, except for the changing task. For example, revenue recognition criteria change from recognition when the invoice is drafted to at delivery, therefore, a new entry processing module is created which contains the same revenue tasks, except for the revenue recognition task, which is different. We illustrate this in Figure 4.9. The impact of the change is not proportional to the number of GAAP used, so no combinatorial effect arises.

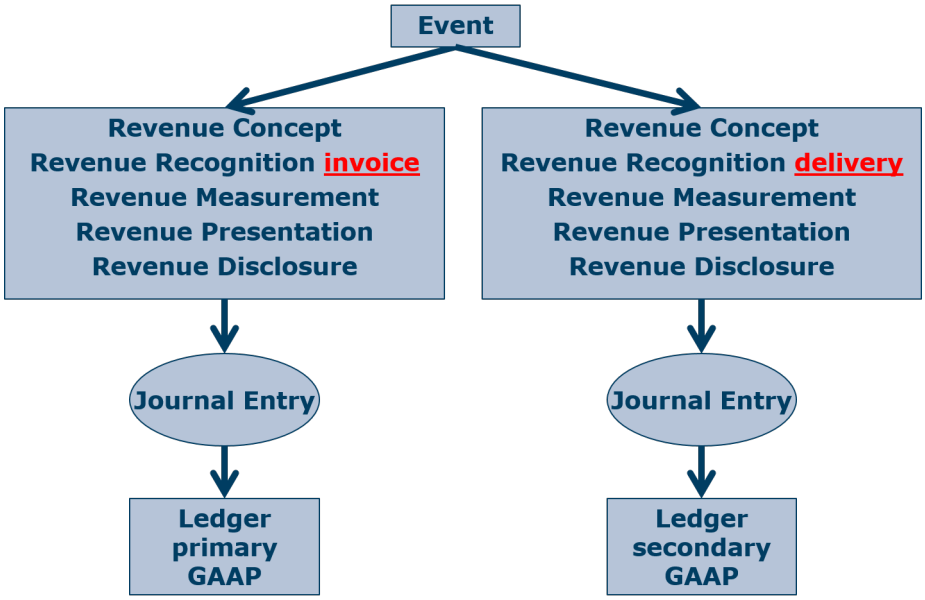

Figure 4.9: Changing recognition criteria when entry processing tasks are not separated

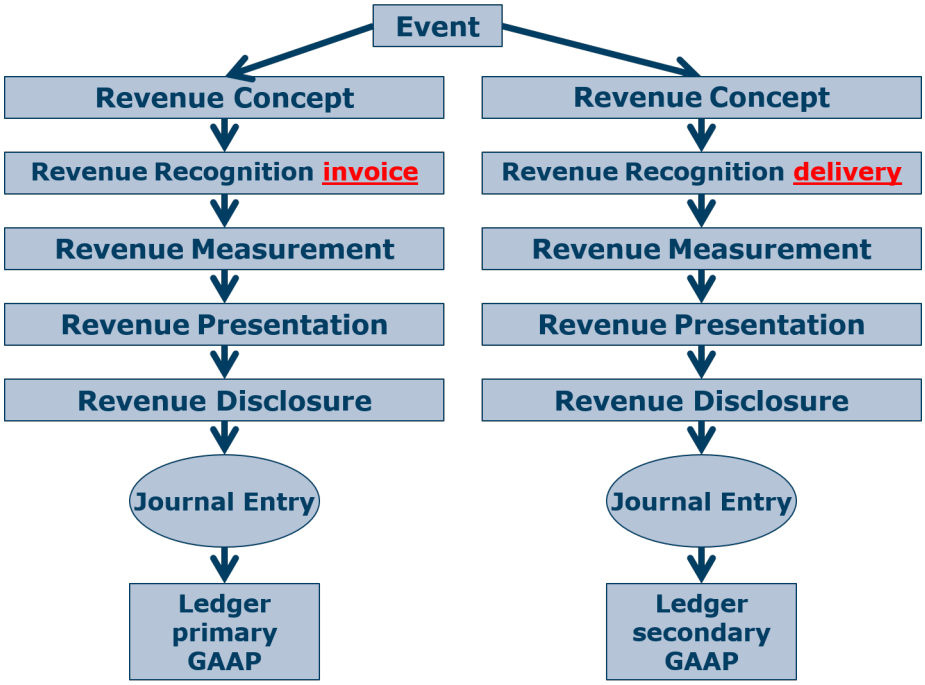

Figure 4.10: Changing recognition criteria when entry processing tasks are separated

4.6.3.2 Context 2: Entry Processing Tasks Separated

Another possible design is separating the entry processing tasks (see Figure 4.8). When the new version of the entry processing task is introduced for the secondary GAAP, a new version of the task needs to be created and other entry processing tasks are duplicated. For example, revenue recognition criteria change from recognition when the invoice is drafted to at delivery. Therefore, a new version of the revenue recognition task needs to be created. We illustrate this in Figure 4.10. The impact of the change is the creation of the new task and duplicating the other entry processing tasks. Since the impact is not proportional to the number of GAAP used, no combinatorial effect is induced.

4.6.3.3 Summary of Impact of Third Change (New Version of an Entry Processing Task for One GAAP - Effect on Entry Processing Module)

In the previous sections, we discussed the impact of creating a new version of an entry processing task for one GAAP when entry processing tasks are separated and when they are not separated. In either case, the change did not cause a combinatorial effect.

4.6.4 Change 4: New Version of an Entry Processing Task for all GAAP

The fourth change we consider, is the impact of a new version of an entry processing task for all GAAP. For example, a company decides to introduce a customer discount on one type of sales, which was never granted in the past. We assume that granting customer discount only requires a change of the revenue measurement method and that both GAAP process customer discount in the same way (post the discount on a separate expense account).

4.6.4.1 Context 1: Entry Processing Tasks not Separated

We use the same context as in change 3 in Section 4.6.3.1 and the accompanying example graphically represented in Figure 4.9: the entry processing tasks (except for recognition) are duplicated in both entry processing modules. When the revenue measurement criteria change, both entry processing modules need to be adjusted. The impact depends on the number of entry processing modules that use those criteria. Otherwise said, the impact depends on the number of differences in entry processing tasks (since tasks are duplicated when a difference occurs) between GAAP. Since the impact is proportional to the number of GAAP (more GAAP results in more differences), the change results in a combinatorial effect.

4.6.4.2 Context 2: Entry Processing Tasks Separated

We use the same context as in change 3 in Section 4.6.3.2 and the accompanying example graphically represented in Figure 4.10: the entry processing tasks (except for recognition) are duplicated for both GAAP. When the revenue measurement criteria change, the impact of the change is proportional to the number of duplicated entry processing tasks for revenue measurement. This number is proportional to the number of GAAP, hence the change causes a combinatorial effect.

4.6.4.3 Summary of Impact of Fourth Change (New Version of an Entry Processing Task for All GAAP)

In the previous sections, we discussed the impact of creating a new version of an entry processing task for all GAAP when entry processing tasks are separated and when they are not separated. In both cases, the change causes a combinatorial effect.

4.6.5 Change 5: Adding a New GAAP to the AIS

The fifth change we consider, is adding a new GAAP to the AIS. For example, a company has an AIS that supports two GAAP (Belgian GAAP and IFRS), but recently merged with an American company and now they also need to provide financial information according to US GAAP.

4.6.5.1 Context 1: Impact on Duplicated Parallel Account Design (Account Design 1a)

When a new GAAP is added when account design 1a is used, all accounts need to be duplicated for the new GAAP. Applied to the modular structure of Figure 4.1 and the extension of Figure 4.2, a whole new set of accounts needs to be created in the chart of accounts. For example, accounts starting with the digits 9, 10, 11 and 12 need to be created for the new GAAP: accounts starting with 9 then are the assets, accounts starting with 10 are equity and liabilities, accounts starting with 11 are expenses and accounts starting with 12 are income. Hence the impact of adding a new GAAP is duplicating all accounts, however, this impact is not proportional to the number of GAAP that are already supported in the system. Therefore, the change does not induce a combinatorial effect.

However, if the limitations of the AIS do not allow to add these accounts, for example, if a fixed number of digits is used for account numbers, the prefix 10, 11 or 12 cannot be used and so adding the new GAAP might require the entire system to be adjusted. In the example, all accounts now need to have an additional digit, and the prefixes for the different GAAP now become 01, 02, 03 and 04 for the primary GAAP and 05, 06, 07 and 08 for the secondary GAAP. This is a combinatorial effect, since not only the accounts of the GAAP that is added need to be duplicated, also all other accounts need to alter

their numbering, hence the impact of the change is proportional to the size of the system. Note that altering the numbering for the accounts might also affect other parts of the AIS that use these accounts, like for example the posting modules.

4.6.5.2 Context 2: Impact on Parallel Account Design using Areas (Account Design 1b)

The impact of adding a new GAAP when account design 1b is used, cannot be readily derived. It is straightforward that for the new GAAP, GAAP-specific accounts need to be created and added to the chart of accounts (also see Figure 4.1 and the extension of Figure 4.3). For example, the accounts of the new GAAP start with the digits 31 for assets, 32 for equity and liabilities, 33 for expenses and 34 for income. But, unless all accounts are created (duplicated), we need to analyze which accounts should be created in the GAAP-specific area. Therefore, an analysis needs to be made which postings are the same for all three GAAP and which are different. Only when a posting is the same for all three GAAP, the posting can be made to the common accounts, otherwise the posting should be made to the GAAP-specific accounts. If more GAAP are already part of the AIS, more work needs to be done to compare postings. Therefore, the impact of the change, being analyzing similarities and differences, is proportional to the size of the system, so the change induces a combinatorial effect.

After the analysis, the GAAP-specific accounts that do not exist yet, need to be created: for each type of posting for which a difference is found, the respective accounts need to be created in the GAAP-specific areas of all GAAP in the system (also see Figure 4.1 and the extension of Figure 4.3). Since the impact of the change is proportional to the size of the system (the number of GAAP), a combinatorial effect is caused. Moreover, also postings need to be adjusted in order to make them use these newly created GAAP-specific accounts. Which is an additional combinatorial effect that is caused.

4.6.5.3 Context 3: Impact on Selection Method 1

The change also has an impact on the selection criteria for reporting. In case account design 1a or 1b is used, only selection method 1 is relevant. The impact of the change is different for the different account designs in combination with the selection method. First, in case account design 1a is combined with the use of selection method 1a, the new GAAP needs to specify the list of accounts to select or add (when using difference posting). However, creating this new list does not have an impact that is related to the size of the system, therefore the change does not inflict a combinatorial effect regarding selection method.

Second, in case account design 1a is combined with the use of selection method 1b, new ranges of accounts need to be specified for reporting in the new GAAP. The impact of this change is not proportional to the size of the system and therefore the change does not cause a combinatorial effect regarding the selection method.

Third, in case account design 1b is combined with the use of selection method 1a, all changes regarding the use of accounts require adjusting the list of *"accounts to include"*. Since the number of changes to accounts depends on the number of GAAP in the AIS, the selection method induces an additional combinatorial effect.

Fourth, in case account design 1b is combined with the use of selection method 1b, the ranges of accounts for the existing GAAP do not need to be updated, since the ranges do not change. The change does require a new range to be set up for the newly added GAAP. But since the impact of this change is not proportional to the size of the system, the change does not inflict an additional combinatorial effect regarding the selection method used.

4.6.5.4 Context 4: Impact on Parallel Ledgers (Account Design 2)

Adding a new GAAP has a limited impact on the parallel ledgers design (account design 2): in the modular structure as depicted in Figure 4.1 and extended by Figure 4.4, a new ledger needs to be created that uses the same chart of accounts as the other ledgers. However, it might be necessary to add accounts to the chart of accounts that are specific for the new GAAP: for example, the impairment of treasury shares is allowed in Belgian GAAP, but not in IFRS. So when the new GAAP requires an account for impairment of treasury shares, this account needs to be created in the chart of accounts. Since the impact of these changes is not proportional to the number of GAAP that are already supported by the AIS, no combinatorial effect is caused by the change.

4.6.5.5 Context 5: Impact on Selection Method 2

In case account design 2 is used, selection method 2 becomes relevant and might cause additional combinatorial effects. Regarding selection method 2a, all accounts of the chart of accounts need to be evaluated whether they are needed for the new GAAP and a new list of exceptions for that GAAP needs to be created. However, the impact of this change is not proportional to the number of GAAP used in the system and therefore, does not cause a combinatorial effect regarding the selection method used.

For selection method 2b, the accounts should also be evaluated and it should be indicated whether they are used by the new GAAP. Since the impact of the change is only proportional to the change and not to the size of the system, the impact does not depend on the number of GAAP already supported by the AIS, the change does not impose an additional combinatorial effect regarding the selection method used.

4.6.5.6 Context 6: Impact on Separate Company Codes Design (Account Design 3)

When using account design 3, the impact of adding a new GAAP depends on whether the same (account design 3b) or a different chart of accounts (account design 3a) is used.

In account design 3c the company codes use the same chart of accounts, but there is a separate chart of accounts for operations and for overhead. Since an account is exclusively present in the operations chart or the overhead chart, the impact is the same as in account design 3b.

When the same chart of accounts is used (account design 3b), the impact of adding a new GAAP (taking into account the modular structure as depicted in Figure 4.1 and the extension of Figure 4.6) is limited to adding a company code to the AIS and linking it to the existing chart of accounts. Since the impact is not proportional to the number of GAAP in the AIS, no combinatorial effect is caused by the change. The same applies to account design 3c as represented in Figure 4.1 and extended by Figure 4.7: the impact is limited to adding two company codes to the AIS and linking them to their respective chart of accounts. Since the impact is not proportional to the number of GAAP in the AIS, no combinatorial effect is caused by the change.

When a different chart of accounts is used by the separate company codes (account design 3a), the impact of adding a new GAAP is not only to create a new company code, but also to create a new chart of accounts for that company code and hereby extending the modular structure of Figure 4.1 and the extension of Figure 4.5 further. Since the impact is not proportional to the number of GAAP in the AIS, no combinatorial effect is induced by the change.

Regarding selection criteria, the impact for account design 3b is similar to the impact in account design 2. For account designs 3a and 3c consolidation is required. Depending on the selection method(s) used in the consolidation description combinatorial effects are induced or not. We already discussed the combinatorial effects caused by selection methods 1a and 2a in the previous Sections 4.6.5.3 and 4.6.5.5.

4.6.5.7 Context 7: Impact on Difference Posting (Posting Design 1)

When adding a new GAAP to the AIS and difference posting (posting design 1) is used, new postings should be added to the system for the new GAAP. All postings for the new GAAP are made by difference with regard to the primary GAAP. The impact of adding these postings is only proportional to the change and is not proportional to the number of GAAP in the AIS (only the difference with respect to the primary GAAP needs to be taken into account). Therefore, adding a new GAAP does not induce a combinatorial effect in case posting design 1 is used.

4.6.5.8 Context 8: Impact on Complete Posting (Posting Design 2)

When adding a new GAAP to the AIS and complete posting (posting design 1) is used, new postings should be added to the system for the new GAAP. All postings for the new GAAP are complete postings, therefore the impact of adding these postings is not proportional to the number of GAAP in the AIS. We conclude that adding a new GAAP does not cause a combinatorial effect in case posting design 2 is used.

4.6.5.9 Context 9: Impact on Posting Design 3

Adding a new GAAP to the AIS does not induce a combinatorial effect when using posting design 3: the impact of the change is that the reverse posting of the posting to the primary GAAP needs to be added to the ledger of the new GAAP, as well as the posting for the new GAAP. Because the impact of this change is not proportional to the number of GAAP in the AIS, but is only proportional to the change itself, the change does not induce a combinatorial effect.

4.6.5.10 Context 10: Impact on Posting Design 4

For posting design 4, adding a new GAAP does not cause a combinatorial effect, since posting design 4 is a mixture of posting design 1 and 2 and neither causes a combinatorial effect. The selection method that is used might induce a combinatorial effect, as described in Sections 4.6.5.3 and 4.6.5.5.

4.6.5.11 Context 11: Entry Processing Tasks not Separated

When the entry processing tasks are all contained in the entry processing module, adding a new GAAP requires to duplicate the entry processing module and its tasks. If needed, some of the tasks can be adjusted according to the requirements of the new GAAP. The change does not inflict a combinatorial effect, since the impact is not proportional to the size of the system (number of GAAP used).

4.6.5.12 Context 12: Entry Processing Tasks Separated

When the entry processing tasks are split into distinct tasks, all separate tasks need to be duplicated for the new GAAP when they are the same for the new GAAP. If the new GAAP requires new entry processing tasks, these should be created separately. However, the impact of the change is not proportional to the number of GAAP in the system and only proportional to the change (adding a new GAAP). Therefore, the change does not induce a combinatorial effect.

4.6.5.13 Summary of Impact of Fifth Change (Adding a New GAAP to the AIS)

In Table 4.5, we provide an overview of the impact of adding a new GAAP to the AIS when using the different account designs, posting designs and the additional impact, depending on the selection method.

Context	Impact of adding a new GAAP	
	CE	No CE
Account design 1a	*	
Account design 1b	x	
Account design 2		x
Account design 3a		x
Account design 3b		x
Account design 3c		x
Selection method 1a	*	
Selection method 1b		x
Selection method 2a		x
Selection method 2b		x
Posting design 1		x
Posting design 2		x
Posting design 3		x
Posting design 4		x
Entry processing tasks:		x
⇒ not separated		x
⇒ separated		x

Table 4.5: Summary table of the impact of change 5: adding a new GAAP to the AIS
∗ represents the occurrence of a combinatorial effect dependent on a specific situation.

4.7 Chapter Conclusions

4.7.1 Conclusion of the Case Studies

In Section 4.4, we describe the multiple GAAP AIS designs from the case studies. These designs have a way to separate postings for different GAAP, we call this design choice the account design. The method used to actually post journal entries to accounts, we call the posting design. Next to these two design choices, also selection methods for reporting and the organization of the entry processing tasks (separated or not) are relevant.

In Section 4.6 we use the FEDS framework (Venable et al., 2014) (we describe our approach in Section 4.5) to evaluate the designs of Section 4.4, with the use of Normalized Systems Theory. We evaluate the impact of the following changes on the different structures:

1. Creating a new account;
2. New version of an entry processing task for one GAAP (effect on journal entries);
3. New version of an entry processing task for one GAAP (effect on entry processing module);
4. New version of an entry processing task for all GAAP;
5. Adding a new GAAP to the AIS.

Context and relation to modular structure	**Change and situation**							
	1	**2**				**3**	**4**	**5**
		A	B	C	D			
Account design 1a: add accounts	x					na.	na.	∗
Account design 1b: add areas and accounts	∗	x	x			na.	na.	x
Account design 2: add ledger						na.	na.	
Account design 3a: add company code, add chart of accounts	x	∗				na.	na.	
Account design 3b: add company code						na.	na.	
Account design 3c: add company codes for overhead and operations						na.	na.	
Selection method 1a: list of accounts	x	na.	na.	na.	na.	na.	na.	∗
Selection method 1b: ranges of accounts		na.	na.	na.	na.	na.	na.	
Selection method 2a: list of exceptions	x	na.	na.	na.	na.	na.	na.	
Selection method 2b: indicate upon creation of the account where to add		na.	na.	na.	na.	na.	na.	
Posting design 1: post by difference	na.	x		x		na.	na.	
Posting design 2: complete posting	na.					na.	na.	
Posting design 3: combine by reverse posting of primary and complete posting for additional GAAP	na.	∗		∗		na.	na.	
Posting design 4: combine difference posting and complete posting	na.	See posting design 1 and 2				na.	na.	
Entry processing tasks:								
⇒ not separated	na.	na.	na.	na.	na.		x	
⇒ separated	na.	na.	na.	na.	na.		x	

Table 4.6: Summary table of the combinatorial effects resulting from changes in the different structures.

Legend of changes
1) Creating a new account
2) New version of an entry processing task for one GAAP - effect on journal entries
3) same as 2) - effect on entry processing module
4) New version of an entry processing task for all GAAP
5) Adding a new GAAP to the AIS

Legend of situations: also see Table 4.3
A) and C) Primary GAAP changes; B) and D) Secondary GAAP changes;
A) and B) Journal entry was the same; C) and D) Journal entry was different.

Legend of symbols
x represents a combinatorial effect in that structure,
∗ represents the occurrence of a combinatorial effect dependent on a specific situation,
na. represents that the structure is not relevant in that change and
an empty cell represents that no combinatorial effect is caused by the change.

We find combinatorial effects in many of these structures, although the manifestation of these combinatorial effects differs. In Table 4.6, we summarize the combinatorial effects resulting from the different changes in the different designs (and we also relate the designs to the modular structure of Figure 4.1). Hence we can conclude that some structures are more evolvable than others. For example, using parallel ledgers (account design 2) avoids the combinatorial effect that arises from using parallel accounts (account design 1). Reporting on these combinatorial effects provides practical proof of evolvability issues in existing multiple GAAP AIS.

4.7.2 Justifying the Problem Statement, Research Gap and Design Objectives

To be able to justify a design science approach for a research project, we need to justify the problem statement, the research gap and the design objectives (Sonnenberg and vom Brocke, 2012). The theoretical justification can be found in the Introduction (Chapter 1) of this dissertation, but in this section we summarize how we provide additional practical justification in this chapter that relates to the problem statement, the research gap and the design objectives.

The problem we study is evolvable multiple GAAP AIS. We discuss the need for evolvable AIS in Sections 1.2 and 2.4. In that way we identify the main **design objective**, being *evolvability*. Further justification for this design objective can be found in Section 1.1: the need for evolvability in the context of multiple GAAP. Moreover, the combinatorial effects from Section 4.6 provide evidence of evolvability issues in current practical multiple GAAP AIS. In Section 1.3, we summarize literature calling for research about multiple GAAP AIS, but none of this literature mentions evolvability, although it is a justified design objective. By electing evolvability as a design objective, we contribute to a **research gap**. In the Introduction (Chapter 1) of this dissertation, we describe the **problem statement** of evolvable multiple GAAP AIS. But in Section 4.1, we raise doubts as to whether the theoretical evidence is enough to justify the problem. Therefore we conduct case studies to research the problem in practice and report on the results in Section 4.4 and Section 4.6. By providing evidence for a lack of evolvability (the combinatorial effects of Section 4.6) in current multiple GAAP AIS, we can further justify the problem at hand.

Combining the need for research stated in literature and the practical evidence, we can state that we justify the problem statement (theoretically and practically), we justify the research gap and the design objectives. Therefore, we believe that the design project of this dissertation is relevant, both for literature and practice.

Chapter 5

Design: Deriving Design Principles for Evolvable AIS that Support Multiple GAAP

In Chapter 4, we identify the problem of evolvable multiple GAAP AIS as being justified and relevant. This adheres to the design science research process of Sonnenberg and vom Brocke (2012). In this chapter, the next activity is described: we design an artifact (the design principles) that contributes to solving the problem (design of an evolvable AIS that supports processing of accounting-relevant events in multiple GAAP). Later, in Chapter 6, we use these design principles to build an evolvable multiple GAAP AIS.

In the design activity, we study the combinatorial effects of Section 4.6 and we establish that some combinatorial effects that occur in one structure (account design, posting design, selection methods and separation of entry processing tasks or not, as described in Section 4.4), do not occur in others. That contrast constitutes the foundation to formulate the design principles. Then we use Normalized Systems Theory to refine and reword the design principles. Next, we reevaluate previous design principles to check whether they are still applicable, do not conflict with the new design principle or whether they should be defined more stringent. This iterative design allows us to adapt and/or refine design principles as we gain more insight into the multiple GAAP problem.

We use the FEDS (Venable et al., 2014) to describe the evaluation phase. First, our evaluation goals are to check to what extend the proposed design principles 1) can be supported by current practice (the case studies) and existing literature and 2) prevent the combinatorial effects described in Section 4.6. Second, the evaluation strategy is a formative (why), ex-ante (when) evaluation that consists of a logical proof (an artificial evaluation) as proposed by Hevner et al. (2004): we relate the design principles to the case studies (also see Chapter 4), as well as to literature (also see Chapter 2). We report on this evaluation directly after introducing the design principle(s). Third, we report on the properties to evaluate: 1) whether case studies already use the design principle and 2) whether the design principle adheres to Normalized Systems Theory and hence when implemented would prevent the combinatorial effect it is supposed to prevent. Fourth, the individual evaluation episodes are reported after each design principle.

In the next sections we use each design choice that needs to be made to design principles that can be derived when comparing the choices. The design choices are: account design, selection method, posting design and separation of entry processing tasks. We will discuss each of these design choices and their impact with regard to each of the changes, we do this using the same template for each design choice:

- Manifestation: we explain the manifestation of the combinatorial effect(s) and we contrast the designs;
- Design principle(s): we propose the actual design principle(s) that should prevent the combinatorial effect(s);
- Evaluation: we relate the design principle(s) to the case studies and to Normalized Systems Theory literature.

5.1 Account Design Choice

Manifestation: Creating an additional account (change 1 of Section 4.6), requires adding a new account in all duplicated sets of accounts in account design 1a (duplicated parallel accounts). A similar combinatorial effect is present in account design 1b when the account needs to be added in the GAAP-specific areas and in account design 3a because the account needs to be added to the different charts of accounts. This combinatorial effect is not present in the parallel ledgers design (account design 2) and the two other separate company codes designs (account design 3b and 3c).

Since the combinatorial effect exists in account designs 1a, 1b and 3a and not in the others, we can state that with regard to evolvability and this combinatorial effect, the use of account design 2, 3b or 3c is a better design choice. This combinatorial effect has two origins: 1) the fact that account design 1a and 1b do not use separate ledgers for the different GAAP and account designs 2, 3b and 3c do and 2) the fact that account design 3a uses a different chart of accounts for the different GAAP and account designs 2, 3b and 3c use the same chart of accounts.

We can conclude that two characteristics of account designs 2, 3b and 3c ensure that the combinatorial effect does not occur in these designs, being 1) they use separate ledgers for different GAAP and 2) they use the same chart of accounts for each GAAP. Important to note is that the combinatorial effect does not manifest itself differently whether the separate ledgers are part of the same company code or not, therefore we use the terminology of *"separate"* ledgers instead of *"parallel"* ledgers. In this way we derive the following two design principles:

First design principle: *journal entries for different GAAP should be posted in separate ledgers.*

Second design principle: *all GAAP should use the same chart of accounts.*

We illustrate the first design principle in Figure 5.1 and the second design principle in Figure 5.2, assuming we extend the modular structure of Figure 4.1.

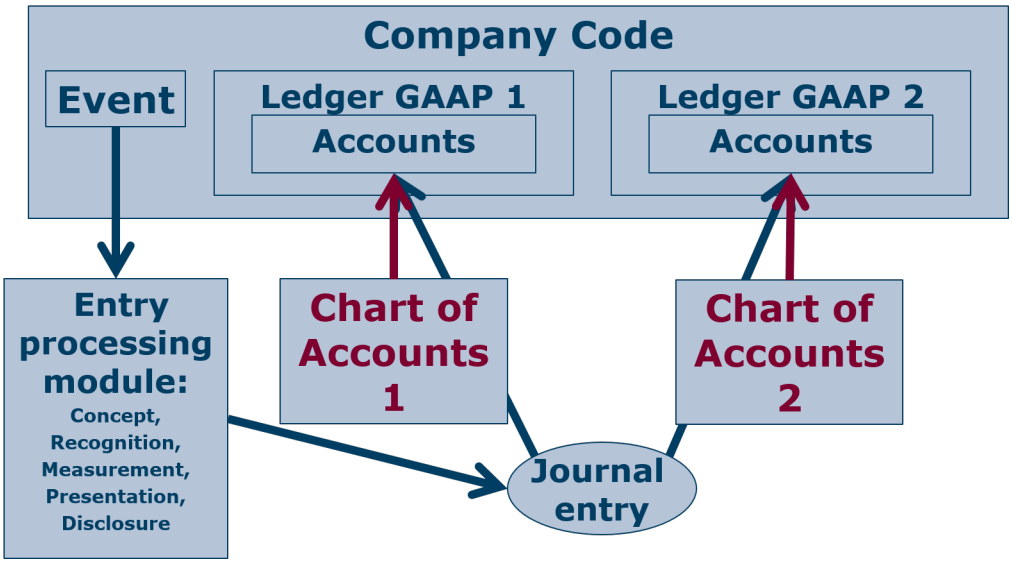

Figure 5.1: Illustration of the first design principle

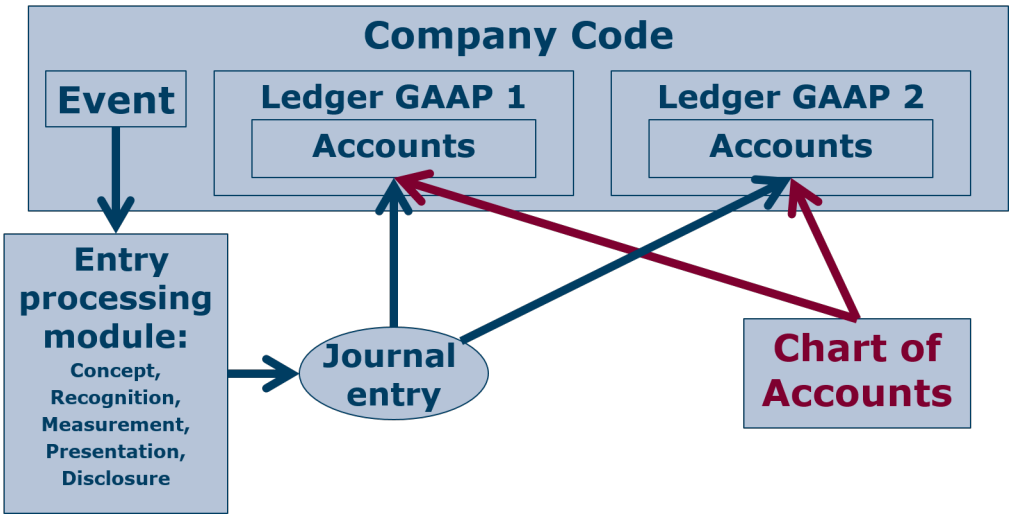

Figure 5.2: Illustration of the second design principle

Evaluation of the First Design Principle: The first design principle is used in three of the five case studies: two of them use a separate company code in their own custom software (account design 3b), the other case study uses account design 3c. We already mentioned the similarity between account design 3b and account design 2 in Section 4.4.1.3. Account design 2 (parallel ledgers) is introduced in version 5.0 of SAP in 2004. Since the market leader in ERP software, SAP, changes its software to be able to support parallel ledgers, it can be considered as an important feature and probably a good practice. The two other cases use account design 1a, because 1) their AIS set-up dates from before the SAP 5.0 update and 2) SAP requires consolidation to add ledgers of different company codes, which requires separate and extensive configuration.

We can also relate the first design principle to the Normalized Systems Theory principles. Following the *"separation of concerns"* theorem, change drivers need to be separated. Since GAAP change independently from each other, they have different change

drivers. Therefore, separating journal entries of different GAAP into separate ledgers is an application of the separation of concerns theorem.

Evaluation of the Second Design Principle: Using the same chart of accounts is compatible with the first design principle: each separate ledger uses the same chart of accounts. This design principle is applied by three of the case studies, namely the ones who use account designs 3b and 3c. Although, in account design 3c two charts of accounts are used, they use different accounts: the same account never occurs twice (one chart contains all the operational accounts, the other all overhead accounts), hence adding a new account does not cause a combinatorial effect, since it only needs to be added in one chart of accounts (operational or overhead).

We can relate this second design principle to the entropy concept from Normalized Systems Theory literature. When an account occurs multiple times (once for each GAAP) in multiple charts of accounts because of the use of account design 3a (separate company codes), the same macro state (account name or number) can refer to multiple micro states (account in each chart of accounts). This can be avoided by applying the second design principle: use the same chart of accounts, so each account (name) only occurs once.

Note that three designs comply with design principles one and two: account design 2, 3b and 3c. However, we should emphasize the differences between the designs which might influence the design choice: in account design 3c configuration of consolidation is required to add the company codes of the different GAAP. In account design 3b, however, this can be configured more easily than in the SAP design, since the custom software of the companies is set-up differently, which resembles more to account design 2. Therefore, which account design is preferred is determined by the selection method design choice, since the other changes (2, 3, 4 and 5 of Section 4.6) do not cause any further combinatorial effects in account design 2, 3b and 3c.

5.2 Selection Method Design Choice

Manifestation: To determine which accounts to select for reporting, we identify four selection methods. After taking into account the first two design principles, only selection methods 2a and 2b are still relevant: selection methods 1a and 1b assume that different charts of accounts are used and it is necessary to define which accounts need to be added. When using the same chart of accounts selection methods 2a and 2b determine how to exclude accounts that are only relevant for one or some of the GAAP used. We refer again to the example of impairment on treasury shares which is allowed in Belgian GAAP, but not in IFRS. In the evaluation of the selection methods in Section 4.6, we identify a combinatorial effect when using selection method 2a (a list of exceptions for each GAAP), that does not occur in selection method 2b (indicate upon creation of an account for which GAAP it is relevant). In that way we can propose the following design principle: *"Upon creation of an account in a Chart of Accounts, it should be indicated which GAAP use the account and which do not."*

Evaluation: In the case studies we found a mixture of selection methods used: two cases use selection method 1a, one uses selection method 2b, one case combines 2b and 1a and in one case we did not obtain information about selection methods. However, we find it difficult to conclude which selection method is better with regard to evolvability: all cases have (other) combinatorial effects in their AIS which makes it difficult to isolate the specific complexity of selection methods. We also did not study the selection methods into enough depth to be able to generalize this principle: the principle remains quite vague and we therefore do not want to propose it as a design principle in our final list.

We can relate the proposed design principle to Normalized Systems Theory's separation of concerns theorem: in this case the change driver is the account which is relevant or not for a specific GAAP. Since the change driver is closely related to the account, the change driver needs to be separated on the level of account and not on another level. For example, taking selection method 2a, the list of *"accounts not to add"* for each GAAP: the list is based on multiple change drivers (multiple accounts that each belong to a GAAP or not), hence a combinatorial effect arises, since the change drivers of the different accounts are mixed. In selection method 2b, the change driver is kept at the level of each account, which prevents the combinatorial effect.

Since this design principle cannot be entirely explained by using the case studies and some implications remain unclear, we hesitate to propose the design principle in our final list. Moreover, since reporting is out of scope of this dissertation (we focus on processing of accounting-related events), we do not further investigate the matter.

5.3 Posting Design Choice

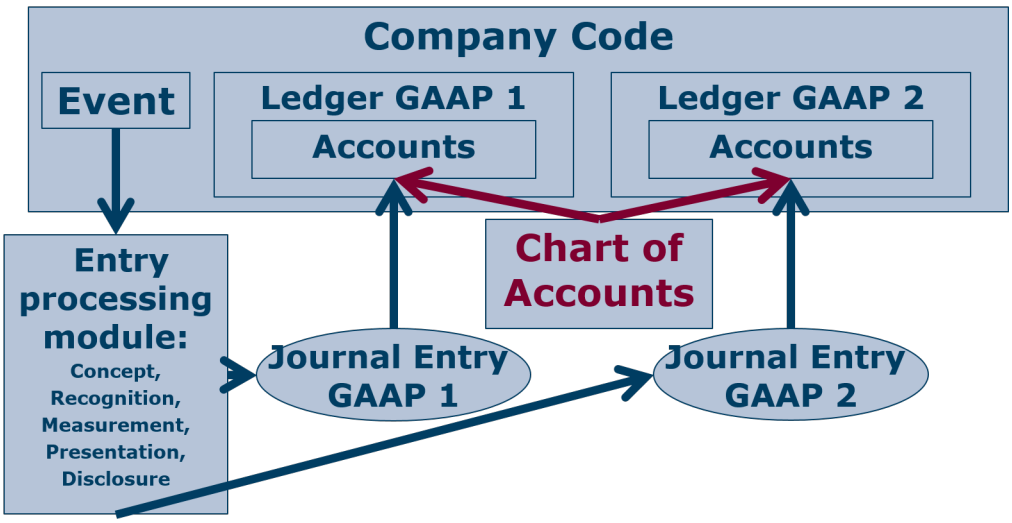

Figure 5.3: Illustration of the third design principle

Manifestation: The first, third and fourth change (of Section 4.6) are not applicable with regard to the posting design. The fifth change does not cause any combinatorial effects in any of the posting designs

However, change 2 (changing entry processing task) causes combinatorial effects in some posting designs. Difference posting (posting design 1) causes a combinatorial effect in situations A and C (Table 4.3), which is not the case when complete posting (posting design 2) is used. This difference occurs because in posting design 2, regardless of the nature or size of the differences or similarities between the GAAP, journal entries are posted independently.

Because posting designs 3 and 4 are a mixture of posting design 1 and 2, a changing entry process task induces a combinatorial effect in some specific circumstances. In this way it becomes clear that these combinatorial effects can only be avoided by using complete postings. Hence, we deduct the following design principle:

Third design principle: *journal entries to different GAAP should be posted independently of each other.*

In Figure 5.3 we illustrate the third design principle by extending Figure 5.2.

Evaluation: The third design principle is not applied in any of the case studies, however, some do use the mixed designs (posting designs 3 and 4). This indicates that the case studies prefer to use complete postings when the difference in accounting treatment between GAAP is too large.

In the insurance company, posting design 3 is used for journal entries regarding financial instruments, since the differences between the GAAP are large: Belgian GAAP requires historical cost to be used for financial instruments, whereas financial instruments in IFRS need to be measured at Fair Value or at amortized cost, which are more complex measurement methods. The use of posting design 4 in the other case study, also implies that when the differences in measurement of (in the case study) fixed assets are large, it is preferred to use independent posting.

This design principle is an application of the first Normalized Systems Theory theorem *"separation of concerns"*. Journal entries that are posted to the ledgers are the result of applying different GAAP. GAAP can change independently, which implies that they have different change drivers. The separation of concerns theorem requires separating change drivers, hence posting journal entries independently is an application of the theorem.

This design principle might give the impression that when there is no difference between GAAP the same work should be done twice and the same journal entry should be duplicated. This is not necessarily true: the design principle suggests that at a conceptual level the postings should be separated, at the implementation level the entry processing module only needs to be carried out once if tasks for the different GAAP are identical and result in only one journal entry, that however than needs to be posted to both ledgers of both GAAP. However, to actually be able to construct software like this, a fine-grained structure is necessary and explicit functionality needs to be added that prevents the duplication.

5.4 Design Choice of Separating Entry Processing Tasks

Manifestation: Whether entry processing tasks are separated or not, does not influence the implications of changes one and two of Section 4.6 and does not cause any combinatorial effect when change five is imposed. Also, changing an entry processing task (change 3 of Section 4.6), does not cause a combinatorial effect on the level of the entry processing module, as presented in Section 4.6.3. It is only when after this change, a second change requires that a common entry processing task (for example, measurement method) needs to change for all GAAP (change 4 of Section 4.6), that a combinatorial effect occurs, regardless of whether entry processing tasks are separated or not. The reason for this combinatorial effect lies in the duplication of certain entry processing tasks: in case that entry processing tasks are not separated, the entire entry processing module is duplicated for different GAAP and in case that entry processing tasks are separated only the entry processing tasks that are the same are duplicated for different GAAP.

In order to prevent this duplication (the module or any of the entry processing tasks), two things need to be taken care of. First, the tasks should be separated from each other instead of being combined in one module. This is because we need to prevent duplication, which we cannot avoid when the tasks are combined into one module and some of the tasks on the inside of the module are duplicated for different GAAP. Second, tasks that are the same for different GAAP should be reused. In this way we prevent that tasks are duplicated and it allows that tasks can change independently. Hence, we deduct the fourth and fifth design principle:

Fourth design principle: *each event that can cause an accounting impact, should be processed by at least five separate tasks (versions of the entry processing tasks: concept, recognition criteria, measurement method, presentation, disclosure) before a journal entry can be posted.*

Fifth design principle: *each entry processing task (concept, recognition criteria, measurement method, presentation, disclosure) that has a separate change driver should be separated in a distinct task, independent of the GAAP.*

In Figure 5.4 we illustrate the fourth design principle and in Figure 5.5 the fifth design principle, by focusing on the entry processing module from the modular structure of Figure 4.1. If we now illustrate the two changes on Figure 5.5, we see that change three of Section 4.6 has caused the revenue recognition task to be duplicated: one version for recognition at invoice date and one version for recognition at delivery. However, when applying the fourth change (of Section 4.6) that requires the revenue measurement method to be adjusted for all GAAP, only one task needs to be adjusted, because the same task is used by both GAAP.

Evaluation of the Fourth Design Principle: Splitting the entry processing tasks is an application of the *"separation of concerns"* theorem from Normalized Systems Theory, since each task has a separate change driver. Moreover, the design principle can be related to the aggregated business process guideline, as presented by Van Nuffel (2011): posting

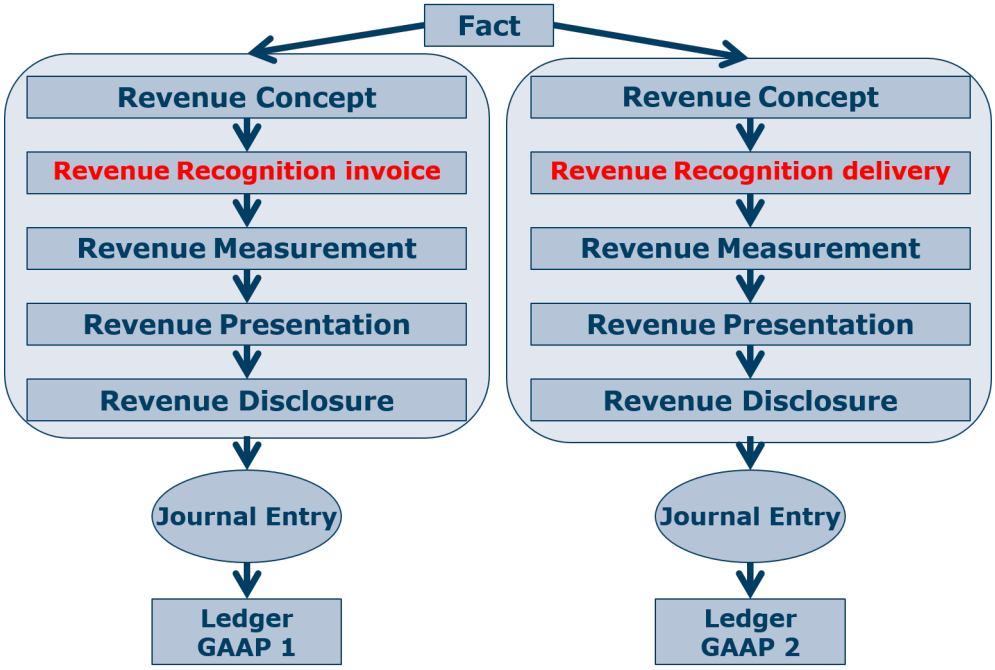

Figure 5.4: Illustration of the fourth design principle

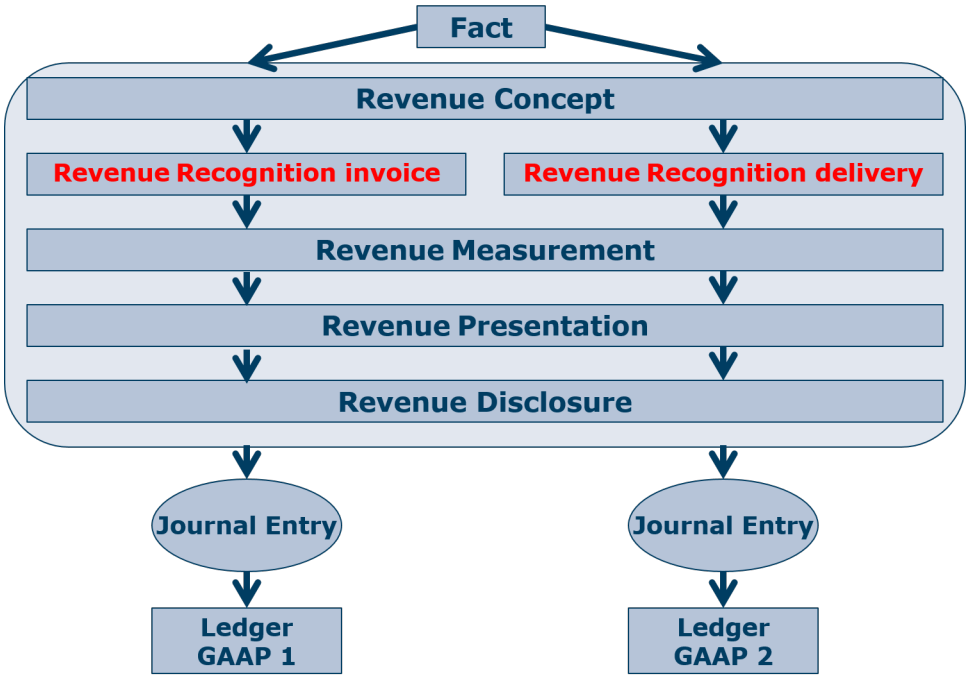

Figure 5.5: Illustration of the fifth design principle

an event in a ledger is the aggregated business process, that consists of (a minimum of) five tasks (the entry processing tasks) at the aggregated level. All these tasks refer to separate, elementary business processes, where the aggregated business process only coordinates the tasks (Van Nuffel, 2011). In this way, a change to one of the tasks does not influence the other tasks or the process as a whole. Each task needs to exhibit version transparency, to prevent that a change in a specific task induces a combinatorial effect in the other tasks or the aggregated business process. For example, when recognition criteria change, a new version of the entry processing task *"recognition criteria"* needs to be created so other tasks do not need to be changed.

Evaluation of the Fifth Design Principle: This design principle can be related to the business rule task of Van Nuffel (2011): each business rule should be separated in a separate task. Applied to the fourth design principle, it denotes that each rule/design principle/criterion related to the definition of concepts, recognition criteria, measurement methods, presentation or disclosure should be separated into a distinct task. These tasks need to be assembled into versions of tasks at a higher level. For example, to process a revenue-related event, we need five related tasks: revenue concept, revenue recognition criteria, revenue measurement method, revenue presentation and revenue disclosure, to be conducted before a journal entry can be posted to a ledger. Each task can have multiple versions, for example the recognition criteria can state that recognition is to be recognized at delivery or at invoice date. At the higher level, depending on which GAAP is used, the right versions of the tasks need to be assembled to process the event, where tasks that are the same for different GAAP are reused. This requires an intermediate level at which these tasks are assembled. Possibly more intermediate levels might be necessary to separate all concerns.

Moreover, we can relate the design principle to the entropy concept: the need to have knowledge of all micro states (the specific combination of fine-grained tasks that are used) that result in a specific macro state (the result of for example, a specific calculation). There is also a need to apply the version transparency theorem to prevent that new versions of tasks cause combinatorial effects. Moreover, the instance traceability theorem is relevant, since it is necessary to know which versions of which tasks are used for a specific journal entry.

5.5 Chapter Conclusions

In this chapter, we use the combinatorial effects of Section 4.6 to develop design principles. For each design choice (account design, selection method, posting design and separating entry processing tasks or not) we provide an overview of the resulting combinatorial effects and we derive design principles. These design principles provide guidance for the design of future evolvable multiple GAAP AIS. We evaluate them theoretically by relating them to the case studies and to Normalized Systems Theory literature. It is important to note that these design principles provide necessary conditions for the design of more evolvable multiple GAAP AIS (if you do not follow the principles the AIS will not be evolvable). However, the design principles do not provide sufficient guidance, since we cannot claim

to have identified all design principles to resolve all combinatorial effects in multiple GAAP AIS that might occur. Five principles are identified, we do not identify a design principle regarding selection methods, because of the indistinctness of its implications. In the next chapter, we provide a practical evaluation of these design principles. The five design principles are:

1. Journal entries for different GAAP should be posted in separate ledgers.
2. All GAAP should use the same chart of accounts.
3. Journal entries to different GAAP should be posted independently of each other.
4. Each event that can cause an accounting impact, should be processed by at least five separate tasks (versions of the entry processing tasks: concept, recognition criteria, measurement method, presentation, disclosure) before a journal entry can be posted.
5. Each entry processing task (concept, recognition criteria, measurement method, presentation, disclosure) that has a separate change driver should be separated in a distinct task, independent of the GAAP.

Chapter 6

Construct: Building an Evolvable Prototype for a Multiple GAAP AIS

In previous chapters we described the problem of evolvable AIS that support multiple GAAP and we proposed design principles that can be used for developing evolvable AIS. We tested these design principles theoretically by relating them to the case studies (Chapter 4) and to Normalized Systems Theory (Section 2.5). In this chapter we evaluate the design principles practically by building a proof-of-concept prototype of an evolvable AIS that supports processing of accounting-relevant events in multiple GAAP. We repeat the research question for this activity from the Introduction (Chapter 1): *What do we learn from implementing the design principles in a proof-of-concept AIS?* This research question can be refined into the following more detailed questions:

- Can we enforce the design principles?
- If not, how can we redefine them?
- What do we need in addition to the design principles to be able to build evolvable multiple GAAP AIS?
- How feasible is it to build evolvable multiple GAAP AIS?

The prototype investigates the relevance and feasibility of the design principles in a realistic setting, which is its sole purpose. The feasibility test means that we want to provide evidence that the design principles can be used to build evolvable multiple GAAP AIS and that the time and effort to do so is reasonable. We also evaluate whether it is possible to build the needed fine-grained structure to obtain evolvability of the AIS. To attain this, we prefer to build a prototype which has a specific, limited amount of functionalities with actual examples (realistic setting). We do this so the design principles can be translated into tangible, concrete design and construction and do not remain vague. The disadvantage of this approach is that the prototype will have limited functionality.

We build the prototype based on examples and we will not conduct an analysis of all needed functionality for an entire multiple GAAP AIS. Of course the skeleton, consisting of the account design, the posting design and the separation of entry processing tasks or

not[1], of a multiple GAAP AIS will be shown in the prototype. However, to be able to extend the functionality of the prototype to support multiple GAAP would require an extensive amount of additional analysis: 1) all possible events that can be registered into the AIS need to be identified 2) a thorough analysis needs to be made on how to record these events, which might reveal a lot of additional combinatorial effects 3) then for all these events the entire analysis needs to be made on how they should be processed in all GAAP that the prototype aims to support 4) this analysis needs to be translated to entry processing tasks and modules that can be executed, so an additional link should be made between the recording of the events and the processing of these events 5) reporting in multiple GAAP requires additional analysis, including an analysis about selection methods. This shows that a considerable amount of additional efforts are needed to build a prototype that can be tested in practice, which is not feasible in the context of this dissertation.

To be able to construct the prototype, we first need to describe the modular structure of Figure 4.1 in terms of the Normalized Systems Theory elements. This includes the identification of data elements, their relationships and their attributes. Next, we use the design principles of Chapter 5 to expand the functionality of the prototype: we introduce each design principle and describe how it affects the design. If we would strictly apply the design science research process of Sonnenberg and vom Brocke (2012), these first steps also belong in the design phase. After the design is specified, we build the prototype, using the Prime Radiant, we describe this process in the remainder of this chapter. It is important to note that the prototype is built iteratively.

At each iteration a new version of the prototype is generated, which we evaluate. The evaluation we conduct adheres to the evaluation framework presented by Venable et al. (2014) in the following ways. First, we explicate the goals of the evaluation. We want to know whether the version of the prototype we have build adheres to the principles of the Normalized Systems Theory and in which way. Moreover, we want to evaluate if the design principles of Chapter 5 are applicable and to which extend they provide enough guidance. Second, we discuss the why, when and how of the evaluation. We use a formative (why), ex-post (when) evaluation, because the final version of the prototype is not a finished product yet and only provides us with insights to further develop AIS. The how of the evaluation is by imposing changes to the prototype and evaluating whether they cause combinatorial effects or not. We use the same changes as in Section 4.6:

1. Creating a new account;
2. New version of an entry processing task for one GAAP (effect on journal entries);
3. New version of an entry processing task for one GAAP (effect on entry processing module);
4. New version of an entry processing task for all GAAP.
5. Adding a new GAAP to the AIS.

[1]We do not consider a selection method design choice, since 1) selection methods are part of reporting functionality and we focus on processing of accounting-related events in multiple GAAP and 2) we did not propose a design principle about selection methods.

In this way we can conclude how we should implement the design principles and what additional implementing principles we need, by relating the conclusions to Normalized Systems Theory principles. Third, the main property we want to evaluate is whether the prototype is evolvable or not. Fourth, the individual evaluation episodes (after each iteration) are described in the remainder of this chapter.

We describe how we build an initial data model and prototype in Section 6.1. Next, we evaluate the initial prototype with regard to its functionality and the first two design principles of Chapter 5, in Section 6.2. Then, we use the first change to evaluate the prototype with regard to evolvability, which we report on in Section 6.3. We extend the functionality of the prototype with regard to posting functionality in Section 6.4 by using the third design principle. Next, we impose the second change on the prototype and evaluate the impact with regard to evolvability in Section 6.5. We use design principles 4 and 5 to build a last version of the prototype in Section 6.6. This last version is evaluated in Sections 6.7, 6.8 and 6.9 by using the third, fourth and fifth change. In the last Section (6.10), we provide examples of how the processing in the prototype works to give the reader a clear idea of what the scope of functionalities is that the prototype supports.

6.1 Building an Initial Data Model and Prototype

In the introduction of this chapter, we explain that the modular structure of AIS (depicted in Figure 4.1) is the starting point of the prototype analysis. This modular structure consists of the concepts event, entry processing module (with five entry processing tasks: concept, recognition, measurement, presentation and disclosure), journal entry, accounts, ledger, chart of accounts and company code. We use the modular structure to identify instances of the Normalized Systems Theory software elements. Moreover, best practices in Normalized Systems Theory literature suggest the identification of these elements on an anthropomorphic basis (i.e., concurring with meaningful real-world entities) (Mannaert et al., 2016). In doing so, we identify the following core *data elements*[2]: Event, JournalEntry, Gaap, CompanyCode, Ledger, ChartOfAccounts and Account.

Several remarks regarding this first set of data elements can be made. First, in this first version of the prototype we leave out the entry processing module and entry processing tasks and focus on the data elements and their relations.

Second, it can be noticed that a many-to-one relationship exists between: Account – ChartOfAccounts, JournalEntry – Ledger, Ledger – Gaap, Ledger – CompanyCode and Event – JournalEntry. A many-to-many relationship exists between JournalEntry – Account, which should be avoided (Mannaert et al., 2016). This leads to the creation of an additional data element JournalEntryLine and a many-to-one relationship between Account – JournalEntryLine and JournalEntryLine – JournalEntry.

Third, the data element Event is too general and should be refined into the different kinds of events that occur and should be recorded in an AIS. We need to do this, to separate

[2]In Section 2.5.4 we explain the concept of data elements and action elements

the concerns of the different events (following the separation of concerns principle from Normalized Systems Theory): for example, selling goods to a customer has different change drivers than the purchase of property (both business-wise as accounting-treatment wise). In a standard order-to-cash process, we can identify Sale and Payment as events following the definition of an event[3] in the REA model. However, the actual process contains far more important steps of which information needs to be kept like for example the order, the delivery and the invoice for a Sale. Since these steps in the process have separate change drivers (order has requirements from the sales force of the company, whereas the invoice is influenced both by regulation and the implementation of that regulation by the accounting department), they should be identified as separate data elements in our prototype. We hence identify SalesInvoice, SalesDelivery and SalesOrder as additional data elements (and we also refer to them as events, although they are technically subevents of the main event Sale).

Lastly, the data element CompanyCode is artificial and non-anthropomorphic (in the sense that it does not concur with an actual real-world concept) in nature. As a consequence, we replace this data element with the data element Company.

Next, *attributes (fields)* for the different data elements can be identified. First, all data elements have an attribute *"name"*, which acts as a unique identifier. Next, for all many-to-one relationships, the data element at the many side of the relationship needs to have an attribute that links an instance of the data element to one particular instance of the other data element. Further, we need the following additional attributes: number for Account, date for JournalEntry, amount for JournalEntryLine and debit/credit for JournalEntryLine. In Figure 6.1, we represent the final model in an entity relationship diagram[4], including the determination of data value types (such as String or Integer).

Based on the data model as described above, we can generate ***a first version of the prototype***. As explained in Section 2.5.5, a set of out-of-the-box functionalities such as CRUD (create, read, update and delete) screens are provided by the code generation as triggered by the Prime Radiant. We can therefore immediately start to create instantiations of the data elements in the model. Stated otherwise: we can start with the configuration of the prototype, being, inserting data of the company for which we want to use this AIS. In this configuration phase the following activities should be carried out: create the Companies (the separate legal entities for which the prototype is used), define the ChartOfAccounts for each Company, create Accounts in the ChartOfAccounts, create the Gaap in which the Company needs to report and create the Ledgers.

This initial data model is rather general and does not yet incorporate the design principles as proposed in Chapter 5. It is not straightforward to actually impose these principles on the system at this point in time. For instance, the first design principle requires to separate ledgers for different GAAP in the system: however, this is not enforced by the data model, it is a choice that needs to be made in the configuration phase of the system. Stated otherwise: the current prototype allows but does not enforce the adherence of the design principles considered. Accurate documentation could assist in this matter. For example, such documentation would describe that the data element Ledger is the

[3]We repeat: *"Economic Event represents either an increment or a decrement in the value of economic resources that are under the control of the enterprise."* (Hruby et al., 2006, p. 4)

[4]In Appendix 1, an explanation of the notation used in the ERD can be found in Figure A.1.

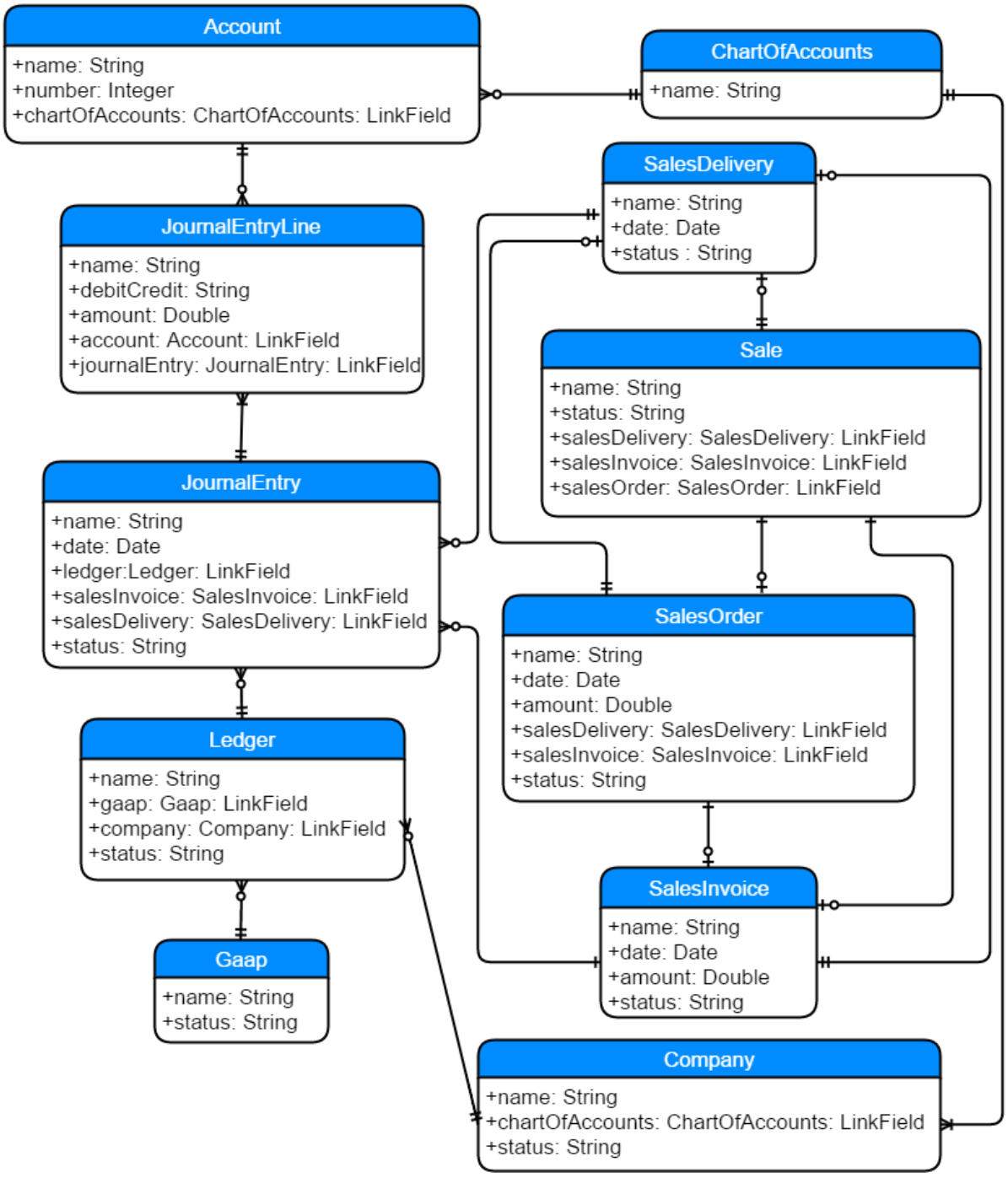

Figure 6.1: Entity Relationship Diagram of the Prototype

collection of JournalEntries according to one GAAP. The second design principle, to use only one chart of accounts, is also not enforced by the current data model and again a configuration setting. Hence, documentation should clarify that all Ledgers (of the same Company) should use the same ChartOfAccounts. Note that this documentation provides guidance at the level of data elements in the actual prototype, which is more specific than the design principles of Chapter 5.

Now we start to configure the prototype for use. First, we create an instance of the data element *Company* for which we want to set up the AIS (we call this company *"Antwerp-Compu"*). Second, we instantiate *"IFRS"* and *"Belgian Gaap"* from the data element Gaap, assuming that the company needs to report according to multiple GAAP. Third, we create a separate *Ledger* for each GAAP (following the design principle *"separate ledgers"*). Next, we create a *ChartOfAccounts* in which all needed accounts can be created (following the design principle *"use the same chart of accounts for all GAAP"*). Finally, we create some *Accounts* that belong to the created ChartOfAccounts (for example, 4700000 Revenue product X and 4710000 Revenue product Y).

6.2 Evaluating the Initial Prototype

In the previous section, we built a first version of the prototype, but we need additional documentation to enforce the first two design principles of Chapter 5, since these were not enforced by the data model itself. Hence, the prototype should ideally contain additional functionality that helps to enforce the design principles. For the first design principle, *"separate ledgers"*, we can add functionality that automatically creates the ledgers needed when a new GAAP or a new company is added to the system. This means we need to define workflows for the data elements Gaap and Company, which we name *GaapFlow* and *CompanyFlow*. These flows are activated after a Gaap or respectively a Company data element is added to the system. Each of these flows consists of two tasks. First a task *CheckLedgers* which verifies which Ledgers need to be added as a consequence of adding the Gaap/Company: if a new Gaap is added, a new Ledger should be created for each Company that uses that Gaap; if a new Company is added, a new Ledger should be created for each Gaap that the Company uses. The second task, *CreateLedger*, is the tasks that actually creates the needed Ledgers.

For the second design principle, *"use the same chart of accounts"*, an additional restriction could be added to the data model: a many-to-one relation between Company and ChartOfAccounts. Further, the prototype should make sure that in a JournalEntryLine only Accounts from the ChartOfAccounts of the Company can be selected. These changes result in a ***second version of the prototype***, which can readily be regenerated by the Prime Radiant.

To further demonstrate the practical value of the design principles, we can impose some changes to the prototype incorporating the design principles of Chapter 5. Analyzing whether these changes result in a combinatorial effect allows us to further evaluate the effectiveness of the design principles and the prototype.

6.3 Evaluating the Impact of the First Change

As a first change, we consider the creation of a new account in the ChartOfAccounts that is used by all GAAP. For example, if the company starts selling a new product, it might want a separate account to record the revenue of the product: account number 4720000 with description *"Revenue product Z"*. This requires the creation of that one Account and the link to which ChartOfAccounts that Account belongs. Since the second design principle requires to use the same ChartOfAccounts for all Ledgers of the same Company, the Account should only be created once within the same Company.

The impact of the change is limited to creating one new account and hence is not related to the number of GAAP, the amount of entries in the system or any other variable reflecting the size of the system. Hence, no combinatorial effect is caused by the change.

By using the prototype, we can therefore conclude that the creation of a new account can be incorporated without a combinatorial effect. The prototype furthermore illustrates the feasibility of the first two design principles of Chapter 5. We can also clearly define the boundary of the current design: the combinatorial effect that arises when adding a new account is prevented as long as the new account is only used by one Company. If a new Account needs to be added to all ChartOfAccounts of all Companies (for example, a new kind of tax on labor is introduced by the government and this new kind of tax needs to be recorded separately in the statement of profit or loss), this change causes a combinatorial effect.

6.4 Adding Posting Functionality to the Prototype

Now that we have set up the structure of our prototype, we want it to be capable to allow the actual posting of journal entries based on events. We use an example in which Revenue needs to be posted after a *SalesInvoice* event. Hence, we create a data element SalesInvoice with a date and amount attribute. The SalesInvoice is related to the data element JournalEntry with a one-to-many relationship.

Furthermore, some processing functionality needs to be added to the prototype because an accounting context requires the processing of a sales invoice resulting in journal entries for all GAAP. We therefore create a *SalesInvoiceProcessor* task element which is triggered after a SalesInvoice is created, that represents an elementary flow element operating on the SalesInvoice data element, we name this flow: *SalesInvoiceFlow*. This task is responsible for processing each individual SalesInvoice which is created by instantiating the appropriate JournalEntries. In a situation with two GAAP having identical recognition criteria, the SalesInvoiceProcessor would create two identical JournalEntries (and their belonging JournalEntryLines).

The SalesInvoiceProcessor only processes everything, additional tasks are needed to actually create the JournalEntries and JournalEntryLines: an additional task *CreateJournal-*

Entry (bridge task[5]) in the *SalesInvoiceFlow* and a new flow for JournalEntry, *JournalEntryFlow*, which has as tasks *CreateJournalEntryLine* (also a bridge task) and *CheckJournalEntry*. We graphically represent this flow in Figure 6.2. Note that we do not add a task to create JournalEntryLines in the Processor flows: creating journal entries needs to be handled by the data element JournalEntry, because it needs to be executed each time a new JournalEntry is created. Therefore, the task *"CreateJournalEntryLine"* is a task of the *JournalEntryFlow*.

The task *CheckJournalEntry* needs to be executed to check whether the JournalEntry is complete or not, for example, this task should check whether the total amount of debited accounts equals the total amount of credited accounts. This results in a ***third version of the prototype***, which can readily be regenerated by the Prime Radiant.

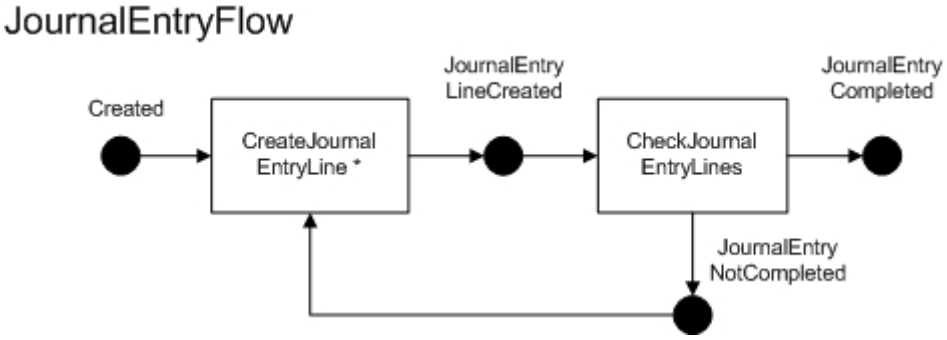

Figure 6.2: Graphical representation of JournalEntryFlow

Table 6.1 provides an overview of the exemplary booking in this context. After the user creates the SalesInvoice with the date 15/08/2016 and amount 20,000, the task *SalesInvoiceProcessor* is executed. The SalesInvoiceProcessor creates two JournalEntries: one is posted to the Belgian GAAP ledger, the other to the IFRS ledger, both on the date of the invoice. In the example, the JournalEntryLines are identical, since there is no difference in accounting treatment between Belgian Gaap and IFRS.

SalesInvoiceProcessor

JournalEntries

Number	Ledger	Date	JournalEntryLine
1	ledger for Belgian GAAP	15/08/2016	link to nr 1 and 2
2	ledger for IFRS	15/08/2016	link to nr 3 and 4

JournalEntryLines

Number	Debit/credit	Amount	Account
1	debit	20,000	Trade Receivables
2	credit	20,000	Revenue
3	debit	20,000	Trade Receivables
4	credit	20,000	Revenue

Table 6.1: Resulting Journal Entry from SalesInvoiceProcessor

[5]A bridge task is a task that merely creates an instance of another data element and is indicated with an $*$ in the figures.

6.5 Evaluating the Impact of the Second Change

As a second change, we study the creation of a new version of an entry processing task for one GAAP. We consider the situation in which revenue recognition criteria change (the same example as in Section 4.6) for IFRS from recognition when the invoice is drafted to recognition when the goods are delivered. For this purpose, we first need to add some data element instances to the prototype to represent this additional complexity. Next to the event SalesInvoice, also the event *SalesDelivery* becomes relevant, therefore we add SalesDelivery to the set of data elements, with date as the only relevant attribute. A SalesDelivery should also be related to a JournalEntry with a one-to-many relationship. Moreover, a relationship needs to be created between SalesDelivery and SalesInvoice.

In practice, a many-to-many relationship between SalesDelivery and SalesInvoice is appropriate. However, if we need to model this, best practices in NST literature (Mannaert et al., 2016) suggest to replace a many-to-many relationship by introducing a new data element that has a one-to-many relationship with the two original data elements. Doing this, also requires additional analysis, since the new data element should correspond to a real-world entity. This analysis is out of scope for this dissertation, since we focus on processing events, not on how to record and assemble them into business processes. There is a need to have a relationship between the two entities, however, for the processing of events it is not relevant what type of relationship exists between the two entities: the identified entities contain the needed information about the event from an accounting perspective, all entities that need to be added because of avoidance of a many-to-many relationship or to model the business process of a Sale do not influence the processing design. Adding a data element between SalesDelivery and SalesInvoice, would increase the complexity of designing and constructing the prototype, without adding insight into the processing of events in multiple GAAP. Therefore, we choose to use a one-to-one relationship between SalesDelivery and SalesInvoice. Of course, we must be aware of functionality that might be added to the business process that does influence the processing design, for example, if after delivery the goods are returned because of a defect, a new subevent needs to be identified (*SalesReturn*), which does have accounting implications (the Revenue can no longer be recognized) and should be taken into account.

To process the SalesDelivery in an accounting context we need a task, *SalesDeliveryProcessor*, to be executed after the SalesDelivery is created. This represents an elementary flow element operating on the SalesDelivery data element, we name this flow: *SalesDeliveryFlow*. The result of the SalesDeliveryProcessor cannot yet be depicted, because at delivery date no instance of SalesInvoice is created yet and the SalesDelivery has no attribute for amount (hence the amount of Revenue to be recognized is not yet known). Therefore, we introduce another event, *SalesOrder*, having the attributes date and amount. SalesOrder also has a one-to-one relationship[6] with SalesDelivery. The amount of the SalesOrder is used as the amount of revenue we recognize on the delivery date. The SalesDeliveryProcessor only processes everything, an additional task is needed to actually create the JournalEntries: an additional task *CreateJournalEntry* (a bridge task) in the *SalesDeliveryFlow* that triggers the *JournalEntryFlow*.

[6]The same as for the relation between SalesInvoice and SalesDelivery: a many-to-many relationship is more realistic, but would increase the complexity of the prototype without any additional insight.

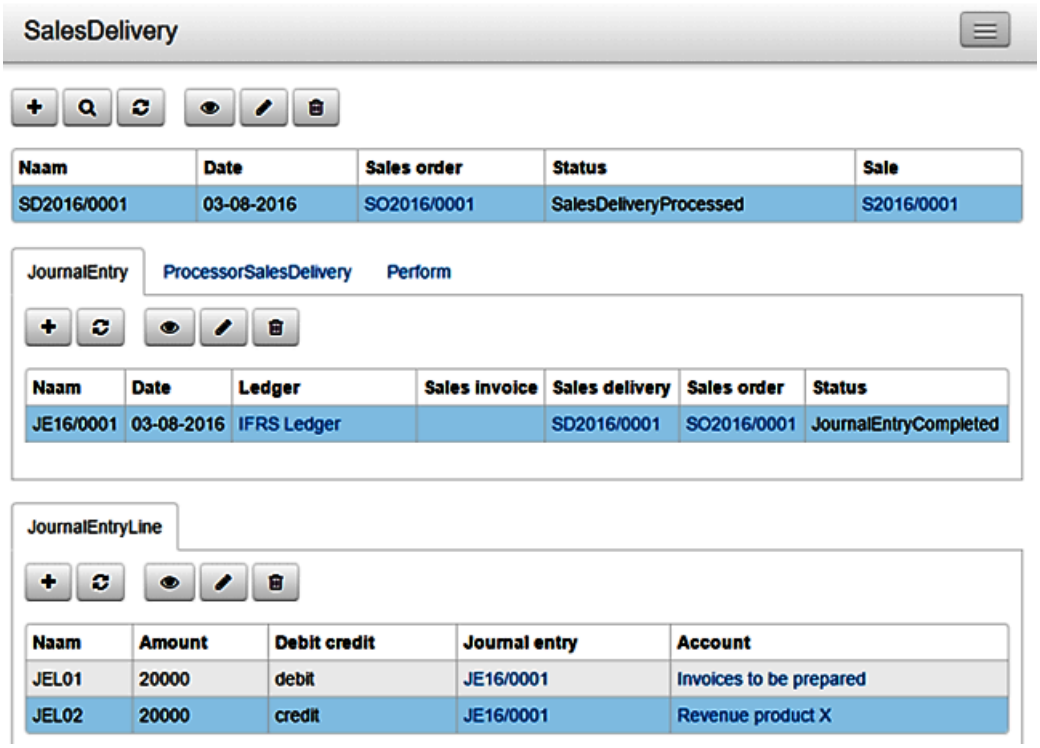

Figure 6.3: First Screenshot of the JournalEntries and JournalEntryLines in the Prototype

SalesDeliveryProcessor and SalesInvoiceProcessor

JournalEntries

Number	Ledger	Date	JournalEntryLine
1	IFRS ledger	03/08/2016	link to nr 1 and 2
2	Belgian GAAP ledger	15/08/2016	link to nr 3 and 4
3	IFRS ledger	15/08/2016	link to nr 5 and 6

JournalEntryLines

Number	Debit/credit	Amount	Account
1	debit	20,000	Invoices to be prepared
2	credit	20,000	Revenue
3	debit	20,000	Trade Receivables
4	credit	20,000	Revenue
5	debit	20,000	Trade Receivables
6	credit	20,000	Invoices to be prepared

Table 6.2: Resulting Journal Entries from SalesDeliveryProcessor and SalesInvoiceProcessor

Now we provide a description of adding data elements into the prototype and describe how these are processed in JournalEntries. First, a SalesOrder is added by a user with date 01/08/2016, amount 20,000 and a unique name SO2016/0001. Nothing further happens. Second, a SalesDelivery is added by a user with date 03/08/2016, a link to the SalesOrder (SO2016/0001) and a unique name SD2016/0001. We show this in Figure 6.3: the first line is entered by the user, being the SalesDelivery. Then, after the SalesDelivery is created, the SalesDeliveryFlow is executed:

- The first task, *"SalesDeliveryProcessor"*, determines that Revenue should be recognized according to IFRS at delivery date. Then the task uses the relationship between SalesOrder and SalesDelivery to look up the amount that should be recognized. The task also determines that the account *"Invoices to be prepared"* should be debited.
- The second task (*"CreatingJournalEntry"*, a bridge task) creates the actual JournalEntry: the details of the JournalEntry can be found in Table 6.2. They are also depicted in Figure 6.3: underneath the SalesDelivery the JournalEntry that is created by the SalesDeliveryProcessor is shown in a waterfall screen: one clicks on the SalesDelivery and sees which JournalEntries are related to the SalesDelivery.
- Moreover, after the JournalEntry is created, the task *"CreateJournalEntryLines"* (a bridge task) in the JournalEntryFlow is executed and creates the JournalEntryLines for the SalesDelivery. The information for these JournalEntryLines originates from the SalesDeliveryProcessor. These JournalEntries are shown in Table 6.2. They are also depicted in Figure 6.3: underneath the JournalEntrys, the JournalEntryLines of the selected JournalEntry are shown.

Third, a SalesInvoice is added by a user with date 15/08/2016, amount 20,000, a link to the SalesOrder (SO2016/0001) and a unique name SI2016/0001. This is shown in Figure 6.4 and Figure 6.5: the first line, the SalesInvoice is entered by the user. After the SalesInvoice is created, the SalesInvoiceFlow is executed:

- The first task, *"SalesInvoiceProcessor"*, just like the SalesDeliveryProcessor, determines all information needed for the JournalEntries. In this case two JournalEntries should be posted: one to the IFRS ledger and one to the Belgian GAAP ledger. These JournalEntries are different, because for IFRS revenue is already recognized on delivery. Therefore, for IFRS the account *"Invoices to be prepared"* is credited and for Belgian GAAP *"Revenue"* is credited. Both debit the account *"Trade Receivables"*.
- The second task, *"CreatingJournalEntry"*, creates the needed JournalEntries: being JournalEntry 2 and 3. The JournalEntries are shown in Table 6.2 and also depicted in Figure 6.4 and Figure 6.5.
- The task *"CreatingJournalEntry"* (a bridge task) initiates the JournalEntryFlow and its task *"CreateJournalEntryLines"*. This task creates the JournalEntryLines 3, 4, 5 and 6 as shown in Table 6.2. JournalEntries 3 and 4 are also shown in Figure 6.4: because in this figure, JournalEntry 2 is selected, the corresponding JournalEntryLines are shown below. In Figure 6.5, JournalEntry 3 is selected and therefore, JournalEntries 5 and 6 are shown below.

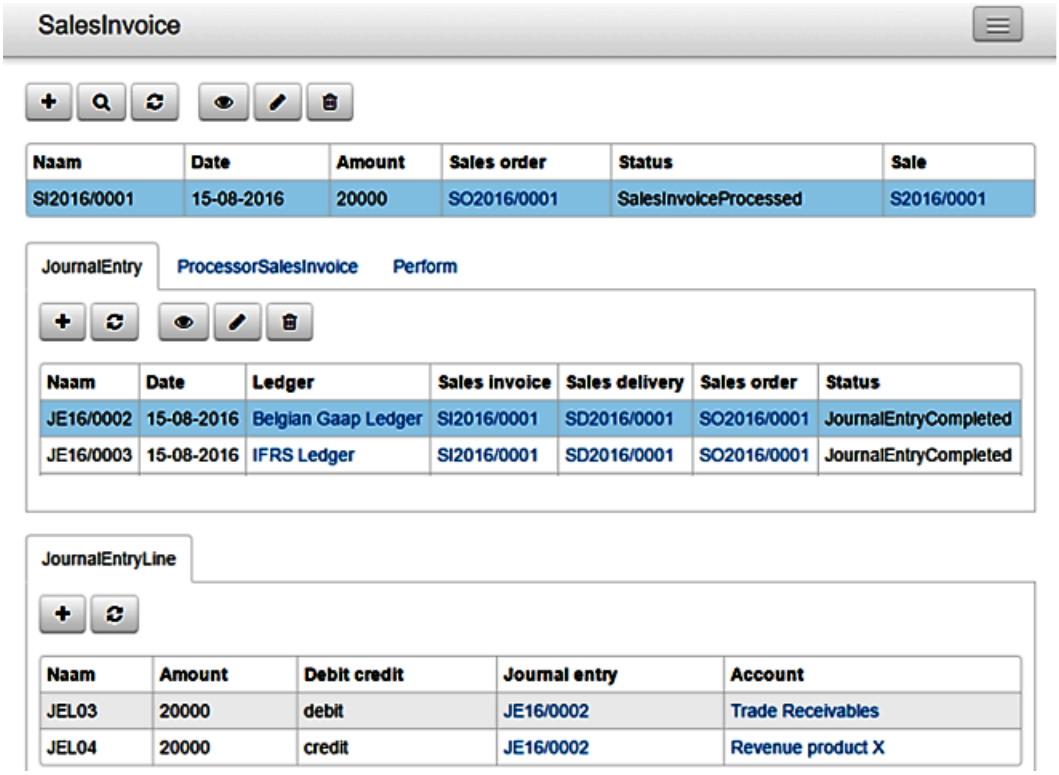

Figure 6.4: Second Screenshot of the JournalEntries and JournalEntryLines in the Prototype

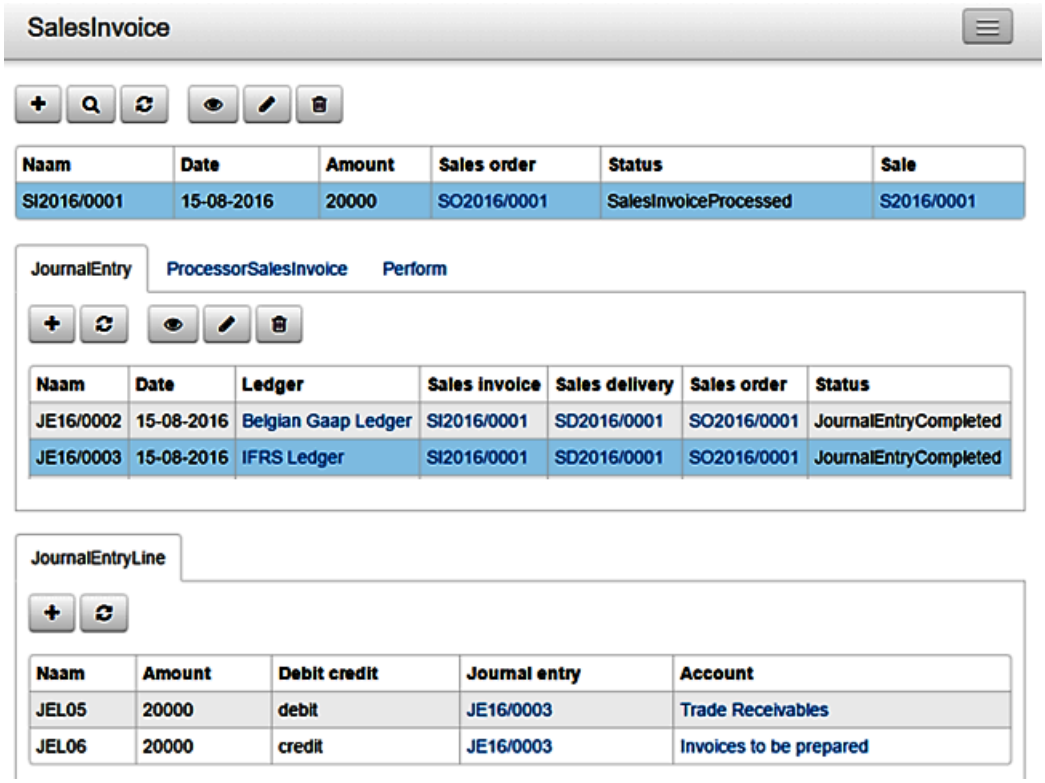

Figure 6.5: Third Screenshot of the JournalEntries and JournalEntryLines in the Prototype

The impact of the change is a change of the JournalEntry for the changing GAAP (IFRS), the creation of a SalesDeliveryProcessor and the additional JournalEntry for IFRS. All these changes are dependent on the change itself and not on the number of GAAP used, the amount of entries in the system or any other variable reflecting the size of the system. We also remark that in the prototype, all functionality is present to automatically process the events that are entered by the user: the SalesDeliveryProcessor and the SalesInvoiceProcessor create all needed JournalEntries and JournalEntryLines.

By using the prototype, we can therefore conclude that changing an entry processing task for one GAAP can be performed without combinatorial effects. The prototype furthermore illustrates the feasibility of the third design principle of Chapter 5: journal entries to different GAAP should be posted independently of each other.

We need to note that following Normalized Systems Theory, the principles should be applied more strict: the processor tasks should be separated for each GAAP. In that way we should create a *SalesDeliveryProcessorIfrs*, *SalesDeliveryProcessorBGaap*, *SalesInvoiceProcessorIfrs* and *SalesInvoiceProcessorBGaap*.

Moreover, we consider the entire entry processing module as one atomic (processor) task for illustrative reasons. In reality the module might consist of different tasks, which are now all executed in the SalesDeliveryProcessor and the SalesInvoiceProcessor. Some more guidance on how to design this inner design of an entry processing module is provided by the fourth and fifth design principles.

6.6 Adding Design Principles Four and Five to the Prototype

In this section we discuss how we add design principles 4 and 5 (of Chapter 5) to a new version of the prototype. Design principle 4 states that each event should at least be processed by the five entry processing tasks before a journal entry can be posted. Therefore, we should create at least five tasks representing the entry processing tasks in the next version of the prototype. In the current version of the prototype, we considered SalesDeliveryProcessorIfrs, SalesDeliveryProcessorBGaap, SalesInvoiceProcessorIfrs and SalesInvoiceProcessorBGaap as task elements that process their respective event, but they do not longer suffice. We need to create workflows that orchestrate the five tasks, being *ProcessorSalesDeliveryBGaapFlow*, *ProcessorSalesDeliveryIfrsFlow*, *ProcessorSalesInvoiceBGaapFlow* and *ProcessorSalesInvoiceIfrsFlow*. In order to do so, we also need to add a data element for each of these flows to the prototype: *ProcessorSalesDeliveryBGaap*, *ProcessorSalesDeliveryIfrs*, *ProcessorSalesInvoiceBGaap* and *ProcessorSalesInvoiceIfrs*. We illustrate one of these flows, the ProcessorSalesInvoiceBGaapFlow, in Figure 6.6.

Designing each flow with its own set of entry processing tasks, violates design principle 5: each entry processing task (concept, recognition criteria, measurement method, presentation, disclosure) that has a separate change driver should be separated in a distinct task, independent of the GAAP. Therefore, we need to determine which tasks are the

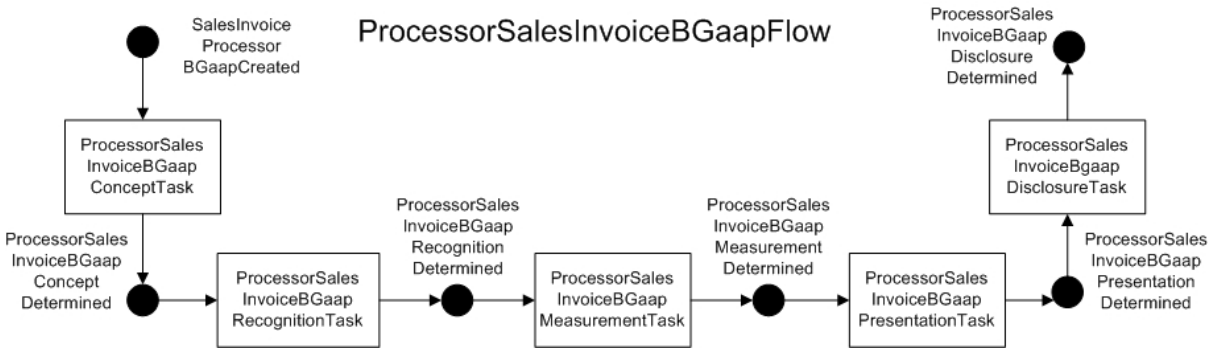

Figure 6.6: Illustration of the SalesInvoiceProcessorFlow containing all five entry processing tasks

same and which tasks are different: tasks that are the same can be reused, tasks that are different can stay different. In the example (we use the same example as in Section 6.5), the recognition criteria for revenue are different for the two GAAP, so we need separate tasks for recognition criteria. For the other tasks we assume there are no differences between GAAP, so they can be reused. We relate this fifth design principle to the business rule task of Van Nuffel (2011) in Section 5.4: each business rule should be separated into a separate task. Applied to the example: each specific rule about one of the five entry processing tasks should be separated as a task. In the following subsections, we zoom in on how we further separate the concerns in the flows illustrated in Figure 6.6.

6.6.1 Determining the Concept: a task at another level

We detect an issue in the first entry processing task, being determining the concept. As mentioned before, each event identified in the REA model can have multiple subevents, however the related accounting concepts are the same for all these subevents. For example, for all subevents related to the REA event *"Sale"*, the accounting concept *"Revenue"* is relevant (in the example this is so for both GAAP). Therefore, it is more logical to execute the entry processing task *"concept"* at the level of the main event, in the example being *"Sale"*. If all subevents, like SalesOrder, SalesDelivery and SalesInvoice, have their own concept task, a combinatorial effect is caused when the concept related to all Sales needs to change. Relating this to Normalized Systems Theory, we can state that the concern of determining the concept, is a concern of the data element of the main event (Sale) and therefore should also be executed there. We identify *Sale* as an additional data element (already included in the ERD model in Figure 6.1) and *DetermineConcept* as a task.

The DetermineConcept task needs to be executed for each GAAP. Following the same reasoning as before, GAAP should be separated from each other, therefore, each GAAP should conduct a separate flow. Hence, two new flows (and their respective data elements[7]) are added to the example: *ProcessorSaleBGaapFlow* and *ProcessorSaleIfrsFlow*. If there would be a difference in concepts related to a GAAP, the task DetermineConcept

[7]When a flow element is created, the Prime Radiant requires that a related data element is also created.

can be used by both flows: the task should consult the business rules in the system to determine which concept is applicable (depending on the GAAP and the event). Therefore, a data element *BRConcept* should be added to the prototype in which all different business rules regarding concepts can be stored. This data element has at least the following attributes: a unique name, the type of event it applies to (for example Sale), the concepts related to the event (for example, revenue and trade receivables) and two dates indicating the period in which the business rule is applicable.

The first entry processing task is now a task (DetermineConcept) executed in the ProcessorSale flows (ProcessorSaleBGaapFlow and ProcessorSaleIfrsFlow). Therefore, the other flows (ProcessorSalesDeliveryBGaapFlow, ProcessorSalesDeliveryIfrsFlow, ProcessorSalesInvoiceBGaapFlow and ProcessorSalesInvoiceIfrsFlow) do not need a task anymore to identify the concept, but they do need a task that checks which concepts are relevant by checking the result of the task (DetermineConcept) of the ProcessorSale flow related to their GAAP. Therefore, we identify a first task for each of these flows, being *CheckConcept*. This task requests from the main event *Sale* which concepts are relevant. Note that the task CheckConcept is the same for each flow.

6.6.2 The Recognition Task that Triggers New Flows

The second task is recognition, however, this task should be executed for each of the concepts that are related to the event. For a Sale, next to Revenue, also Trade Receivables is a relevant concept. Therefore, the second task of the ProcessorSalesDeliveryBGaapFlow, ProcessorSalesDeliveryIfrsFlow, ProcessorSalesInvoiceBGaapFlow and ProcessorSalesInvoiceIfrsFlow, is *"CreateConceptFlows"*, this task creates a new data element for each of the relevant concepts (in the example, *"TradeReceivablesConcept"* and *"RevenueConcept"*) and therefore, triggers the execution of the flows of these concepts. Each of these flows executes the remainder of the entry processing tasks.

The next task (*"CheckConceptFlows"*) in the flow of the Processors (ProcessorSalesDeliveryBGaapFlow, ProcessorSalesDeliveryIfrsFlow, ProcessorSalesInvoiceBGaapFlow and ProcessorSalesInvoiceIfrsFlow) then is to check whether the triggered conceptflows (in the example, *"TradeReceivablesConceptFlow"* and *"RevenueConceptFlow"*) have reached their ending status.

Next, the task *"CreateJournalEntry"* is executed. A JournalEntry can be created, because during the execution of the different tasks in the flow, the data element relating to the flow stores all relevant information that is needed to create the JournalEntry (and related JournalEntryLines).

The last task of the Processor flows checks whether all needed JournalEntries are created. If so the ending status *"JournalEntriesCompleted"* is reached, otherwise the status becomes *"JournalEntriesNotCompleted"* and additional JournalEntries are created. We graphically represent an example of the renewed Processor flows, the ProcessorSalesInvoiceBGaapFlow, in Figure 6.7.

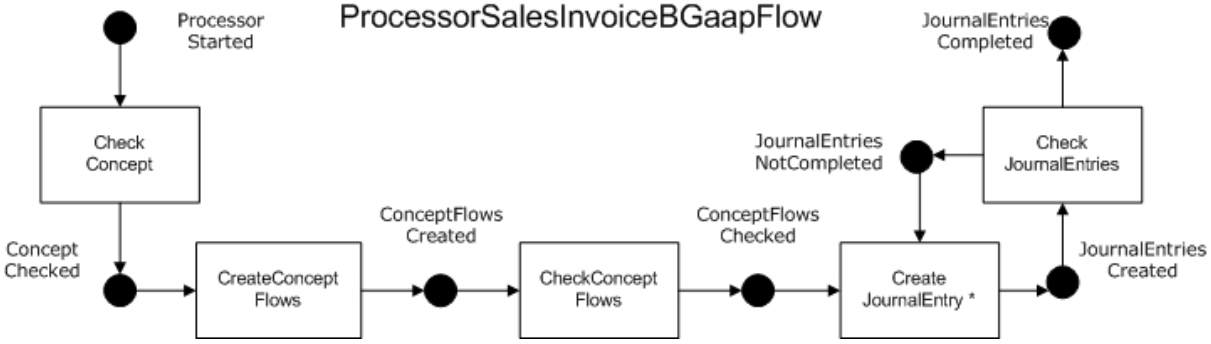

Figure 6.7: Example of a workflow containing entry processing tasks: ProcessorSalesInvoiceBGaapFlow

6.6.3 The RevenueConceptFlow and the TradeReceivablesConceptFlow

The concept flows (in the example, the *RevenueConceptFlow* and the *TradeReceivablesConceptFlow*) are initiated by the Processor flows: they create a data element of the concept type that triggers the execution of the flow.

The first task in these concept flows is the recognition entry processing task. In the example, revenue recognition is different for the two GAAP. To model this, we should use business rules, following the business rule task of Van Nuffel (2011). We hence create a data element *BRRecognition* with at at least the following attributes: a unique name, the concept(s)to which the rule applies (for example, revenue and trade receivables), the event in which recognition should take place and two dates indicating the period in which the business rule is applicable. In the example two business rules are created: *RevenueRecognitionDelivery* and *RevenueRecognitionInvoice*.

To consult these business rules, a separate task needs to be created, being *CheckRecognitionBusinessRules*. This task is used by all different *"concept"* flows, the result depends on the GAAP used. If none of the business rules is applicable, the flow has an alternative path. For example, no JournalEntries for revenue need to be made on delivery because for that GAAP the business rule is to recognize revenue at invoice date. It can be that disclosures are still relevant, even though no JournalEntry should be made. The outcome of the *CheckRecognitionBusinessRules* task hence can have two end states: BRRecognitionFulfilled or BRRecognitionNotFulfilled. Depending on the state, the flow continues to the next recognition task or to the disclosure task.

The task CheckRecognitionBusinessRules only checks the business rules. Therefore, a task to apply the business rules is needed: *RevenueRecognition* for the *"RevenueConceptFlow"* and *TradeReceivablesRecognition* for the *"TradeReceivablesConceptFlow"*. This task determines, the date and Ledger of the JournalEntry and the different JournalEntryLines with the respective Accounts and debit/credit attribute. In this version of the prototype, we consider this all part of one task, however, in future extensions it might be necessary to use multiple tasks to be able to separate all concerns related to recognition.

The next entry processing task is measurement, for which we can use an analogous rea-

soning as for recognition. We first need to consult which business rules are applicable for measurement. Therefore, we create a data element *BRMeasurement* and the task *CheckMeasurementBusinessRules*. Next, the implementation of measurement requires another task, which we call *RevenueMeasurement* and *TradeReceivablesMeasurement*. This task determines the amount of the JournalEntryLines. Since these two tasks are the same for each flow, we can reuse them. In the same way, we create business rules related to the last two entry processing tasks, *BRPresentation* and *BRDisclosure* and the related tasks, *CheckPresentationBusinessRules*, *RevenuePresentation*, *TradeReceivablesPresentation*, *CheckDisclosureBusinessRules*, *RevenueDisclosure* and *TradeReceivablesDisclosure*. These tasks are also reused by all flows.

In Figure 6.8 we illustrate one of the flows, the RevenueConceptFlow, containing all tasks. The TradeReceivablesConceptFlow is executed in a similar way. Adding these new functional requirements results in a ***fourth (and last) version of the prototype***.

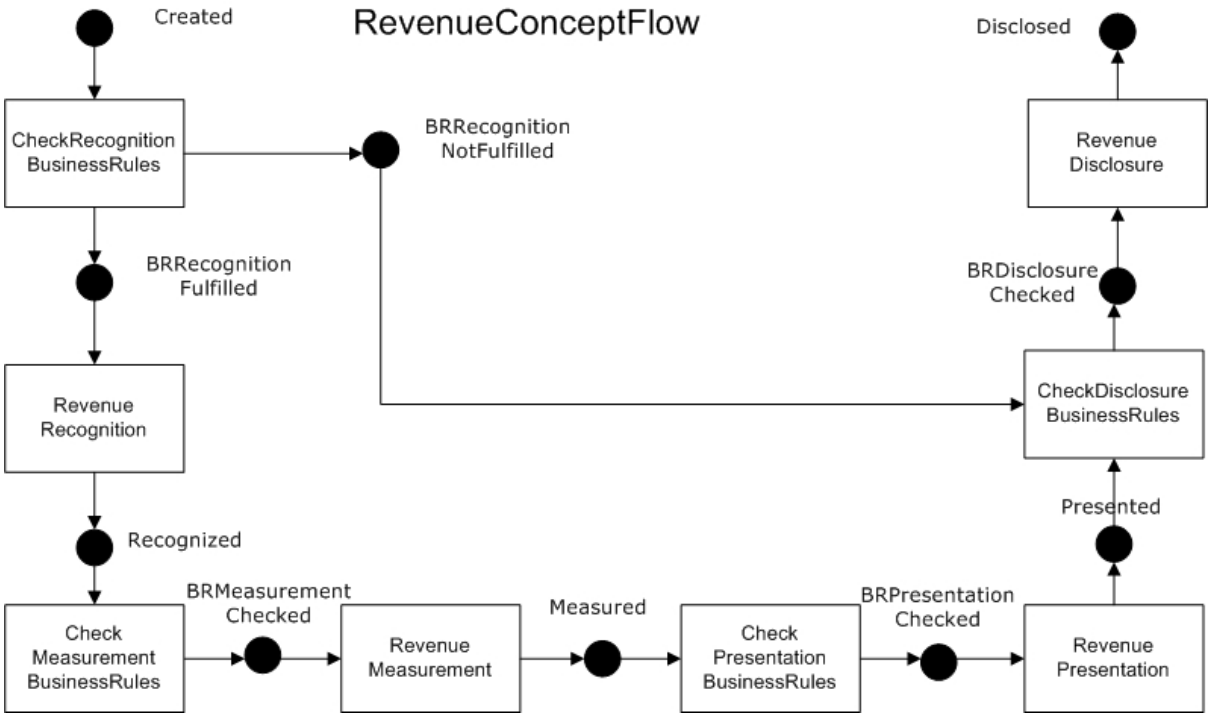

Figure 6.8: Example of a concept workflow containing entry processing tasks: RevenueConceptFlow

6.6.4 Concluding remarks on adding the Fourth and Fifth Design Principle

We should remark that the concept flows RevenueConceptFlow and TradeReceivablesConceptFlow are used for each GAAP that uses the same concepts, however the specific tasks might differ. For example, because the GAAP have other business rules for recognition, the flow takes a different route (recognition criteria fulfilled or not). This is illustrated in Section 6.10.

We also remark that the four processor flows (ProcessorSalesDeliveryBGaapFlow, ProcessorSalesDeliveryIfrsFlow, ProcessorSalesInvoiceBGaapFlow and ProcessorSalesInvoiceIfrsFlow) as well as the concept flows (RevenueConceptFlow and TradeReceivablesConceptFlow) are aggregated business processes, following the guideline of Van Nuffel (2011). The function of the flows is to coordinate all tasks related to processing the event according to a particular GAAP.

6.7 Evaluating the Impact of the Third Change

The third change we study, is the creation of a new version of an entry processing task for one GAAP. Like in Section 4.6, we use revenue recognition criteria that change as example. As this version of the prototype already supports two different sets of recognition criteria, we discuss the impact of changing the revenue recognition criteria of Belgian GAAP from recognition at invoice date, to recognition at delivery date.

This change has no impact on the design of the prototype as we described in the previous section. The only change is that the business rule RevenueRecognitionInvoice becomes obsolete and the business rule RevenueRecognitionDelivery now needs to be used when revenue is recognized according to Belgian GAAP. This implies that in the RevenueConceptFlow that is created by the ProcessorSalesInvoiceBGaapFlow the CheckRecognitionBusinessRules now returns *"BRRegognitionNotFulfilled"* and for the ProcessorSalesDeliveryBGaapFlow the new business rule should be found and applied. That requires that the new business rule is linked to the changed GAAP and the old business rule expires as from a certain date.

Consider another example: (the very unlikely situation) changing revenue recognition criteria for Belgian GAAP from recognition at invoice date to recognition at order date. Because *SalesOrder* has no Flows defined in the current model yet, neither does it have a relation with JournalEntry, the impact of the change is larger than in the previous case: defining a relation between the JournalEntry and SalesOrder data elements, creating a flow *ProcessorSalesOrderBGaapFlow*, creating the actual workflow by assigning tasks to the new flow element and creating a new business rule *RevenueRecognitionOrder*. However, the impact of the change is only proportional to the change and not to the number of Gaap or any other variable representing the size of the system, hence we can state that no combinatorial effect is induced.

6.8 Evaluating the Impact of the Fourth Change

As a fourth change in Section 4.6, we consider the effect of a new version of an entry processing task for all GAAP. For example, changing measurement criteria: a company decides to introduce a customer discount on one type of sales, when this was never granted in the past. We assume that granting customer discount only requires a change of the

revenue measurement method and that both GAAP process customer discounts in the same way (post the discount on a separate account[8]).

What is important to note is that this change, as described in Section 4.6, only causes a combinatorial effect when there is already a difference between GAAP for another entry processing task. Therefore, it is important that we apply this change in the prototype in a similar case. We use the same example as before, but we consider the version of the prototype at the end of Section 6.6 as a starting point: there is a difference between Belgian GAAP (recognition at invoice date) and IFRS (recognition at delivery date) regarding revenue recognition criteria.

Applying the change requires us to adjust the system in a few ways. The amount of customer discount needs to be added to the system, so the SalesOrder and SalesInvoice data element need an additional attribute *customerDiscount*. Next, we need to update the measurement criteria. There are two tasks related to revenue measurement: CheckMeasurementBusinessRules and RevenueMeasurement. A new business rule should be created that specifies that a customer discount needs to be recorded on a separate account. This new business rule does not require the CheckMeasurementBusinessRules to be updated, however the flows using this task now hit upon a business rule regarding measurement which was not the case before the change. Adding the business rule is not sufficient, since the business rule needs to be applied, the task RevenueMeasurement should also be updated. In practice, the task RevenueMeasurement consists of multiple sub-tasks, so we need to add a sub-task that handles the customer discount. The new version of the RevenueMeasurement task is used by all RevenueConceptFlows for all GAAP.

The impact of the change is therefore only dependent on the change itself and not on the size of the system: independently of how many GAAP use the RevenueMeasurement task, the effort of updating the task remains the same. Therefore, the change does not impose a combinatorial effect on the prototype.

6.9 Evaluating the Impact of the Fifth Change

The fifth change in Section 4.6 is adding a new GAAP to the AIS. As an example, we use adding US GAAP to an AIS that already supports IFRS and Belgian GAAP. To apply the change in the prototype, we start by adding a new instantiation of the data element Gaap to the prototype. When a new Gaap is created, the prototype promts us to also create a new Ledger for the new Gaap. The new Gaap will automatically make use of the chart of accounts that is used in the other ledgers for this Company.

To be able to make postings for the new Gaap, we need to create flows for all data elements: *ProcessorSaleUsGaapFlow*, *ProcessorSalesOrderUsGaapFlow*, *ProcessorSalesDeliveryUsGaapFlow*, and *ProcessorSalesInvoiceUsGaapFlow* in the example. Next, the flows should be configured by reusing the tasks that are the same as for the other Gaap and creating new tasks if the task is different from the tasks of the other Gaap already

[8]We consider here IFRS rules that require revenue to be calculated as the total amount reduced with the discount amount.

in the system. The same goes for business rules: when the new Gaap uses other business rules for processing its events (in the example a Sale), they need to be added to the prototype.

Creating all the flows is an effort that is proportional to the size of the system (the number of events and subevents in the system), therefore it can be considered a combinatorial effect resulting from adding the new GAAP to the system. However, this effort can be automated: when a new GAAP is added to the system, all flows for events are automatically created. Next, you could argue that the fact that all these flows need to be configured is also a combinatorial effect, we however believe that this is a logical consequence that is related to the change itself: by adding a new GAAP, the new GAAP needs to process all events accounting wise. Configuring the flows is related to the core of the change: introducing the new GAAP. The prototype can accommodate this by providing the possibility of reusing all flows from another GAAP that is already configured in the system, so the user only needs to adjust the flows in case of differences between the new GAAP and the GAAP from which the flows were copied.

6.10 Illustration of the Functionality of the Prototype

In this section we show what the current scope and supported functionalities of the prototype are. After the prototype is generated, it should first be configured. In the last paragraph of Section 6.1, we describe that the needed Company, Gaap, Ledger(s), Chart(s)OfAccounts and Account(s) that the user wishes, need to be entered into the graphical user interface of the prototype. We also assume that the flows we describe in the remainder of the sections, are configured in the prototype and the needed tasks are programmed with custom code to provide the needed functionality.

Once this is all set-up, the user can start using the prototype and (manually) enter events and/or subevents that need to be processed. The current version of the prototype is limited to support the event Sale and all related subevents that we currently identified: SalesInvoice, SalesDelivery and SalesOrder. For these events all relevant tasks and flows (and hence the needed additional data elements) are created. After the event is created, the flow that is related to that event is executed and automatically creates all related journal entries and needed journal entry lines. In the following subsections, we discuss each step of the different flows into more detail to give an idea of the inner working of the prototype. We do this by using the same example we use throughout this entire dissertation: a sale of 20,000, the order for this sale occurs at 01/08/2016, the delivery at 03/08/2016 and the invoice is drafted and send at 15/08/2016.

6.10.1 The Configuration Phase

As described in Section 6.1, the following instances of data elements are set-up for use:

- Company: "AntwerpCompu"
- Gaap: "IFRS" and "Belgian Gaap"
- Ledger: "IFRS ledger" and "Belgian Gaap ledger"
- ChartOfAccounts: "AntwerpCompu Chart"
- Account:
  - 4700000 Revenue product X
  - 4710000 Revenue product Y
  - 1400000 Trade Receivables
  - 1404100 Invoices to be prepared
- Business Rules:
  - BRConcept: "SaleConcept"
  - BRRecognition:
    - RevenueRecognitionDelivery
    - RevenueRecognitionInvoice
    - RevenueRecognitionOrder
    - TradeReceivablesRecognitionInvoice
  - BRMeasurement:
    - RevenueMeasurement
    - TradeReceivablesMeasurement

6.10.2 The SaleFlow

To enter a Sale in the prototype only a name and a status need to be provided. The name can be chosen by the user, or can be generated automatically as a primary key that is used as a unique identifier for the Sale. The status is important, because the prototype is set-up in such a way that the status of a certain data element determines which task (in its flow) is executed. Ideally, the first status should also be generated automatically, but currently this is not yet supported. The logical first status would be *"created"* to indicate that a new data element of that kind is entered by a user. The status *"created"* triggers the execution of the SaleFlow. However, as we discuss in Section 6.6, multiple flows need to be triggered to separate the concern of the different GAAP. Therefore, a cascade of different flows is triggered after a Sale is created: we graphically represent this in Figure 6.9.

First, the SaleFlow is executed and its first task creates a new data element of the type *"ProcessorSale"* (this is a bridge task, indicated with an $*$ in the figure), which triggers the second flow *"ProcessorSaleFlow"*. The second task in the SaleFlow checks whether its subflow, in this case being the ProcessorSaleFlow has reached its final status, being *"ProcessorSaleProcessed"*. Only then the second task will continue to the next status for the SaleFlow, being *"SaleProcessed"* to indicate that the SaleFlow has reached its ending status. For the user this is a check to see whether the Sale is processed as should be done by the related processors or not: when the status of a certain Sale is

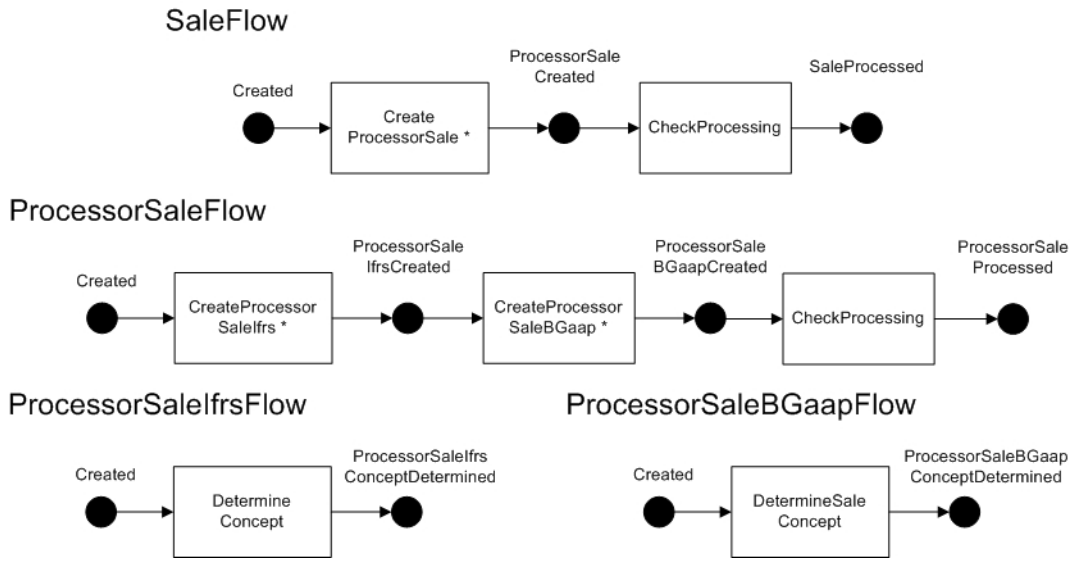

Figure 6.9: The SaleFlow and its related processing flows

"ProcessorSaleCreated", the user knows that the processor is created but not all flows are successfully executed yet, when the status of a certain Sale is *"SaleProcessed"* the user knows that all needed processing activities are finalized.

The second flow that is triggered is the *"ProcessorSaleFlow"*. This flow merely creates processor elements for each GAAP that is needed, in the current example being Belgian Gaap and IFRS. If however, the prototype would need to support US GAAP as well, an additional task would be inserted in this flow, which triggers the creation of an additional data element and its related flow (we described this in Section 6.9). The last task is again a task that merely checks whether the created flows (*"ProcessorSaleIfrsFlow"* and *"ProcessorSaleBGaapFlow"*) reached their ending status.

The *"ProcessorSaleIfrsFlow"* and *"ProcessorSaleBGaapFlow"* both have only one task that is executed, being *"DetermineConcept"*. This task determines that the related concepts for Sale are *"Revenue"* and *"Trade Receivables"* and stores this information in their related data elements *"ProcessorSaleIfrs"* and *"ProcessorSaleBGaap"*. In this example, this task is the same for both GAAP and therefore is a task that can be reused. If both flows have reached their ending status *"ProcessorSaleIfrsConceptDetermined"* and *"ProcessorSaleBGaapConceptDetermined"*, also the above flows can reach their ending states (because the check tasks will respectively see that the end status is reached).

6.10.3 The SalesOrderFlow

After entering a new Sale, a user probably immediately creates a SalesOrder as well. To insert a SalesOrder, a name, a date and an amount should be provided as well as a link to the related Sale. This link can be automated if the user uses pre-defined waterfall screens: after entering a Sale data element and selecting it, a waterfall screen appears in which related data children (being the data elements with which the data element has a one-to-many relationship, in this case SalesOrder, SalesInvoice and SalesDelivery) of the

data element can be added. If a new data element SalesOrder is created by using this waterfall screen, the link to selected Sale is automatically filled in into the input screen. We provide a screenshot of this functionality in Figures 6.10 and 6.11.

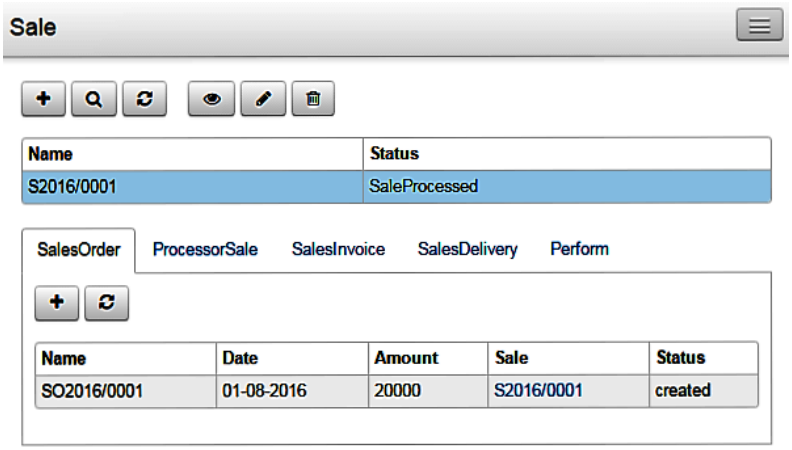

Figure 6.10: Waterfall screen of Sale

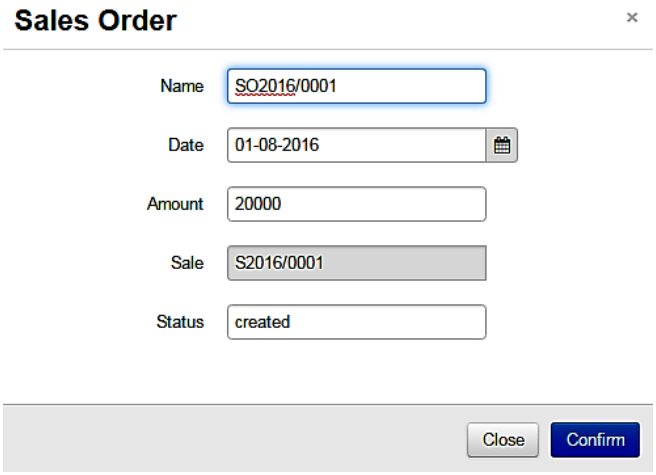

Figure 6.11: Creation of SalesOrder using the waterfall screen: Sale is automatically filled in (the box has a grey background)

SalesOrder also requires a status, the first status we use is *"created"*. This status triggers the *"SalesOrderFlow"* to be executed. Just like for Sale, this flow creates another data element, *"ProcessorSalesOrder"* and checks whether the created data element has reached its ending status. This data element triggers its own flow, *"ProcessorSalesOrderFlow"*, which creates two other data elements, *"ProcessorSalesOrderIfrs"* and *"ProcessorSalesOrderBGaap"* and checks whether the related flows reached their ending status. We graphically represent these first two flows in Figure 6.12. The *"ProcessorSalesOrderIfrsFlow"* and *"ProcessorSalesOrderBGaapFlow"* have the same tasks and states as the ProcessorSalesInvoiceBGaapFlow in Figure 6.7. In the next subsections, we discuss each of the tasks of the two flows, *"ProcessorSalesOrderIfrsFlow"* and *"ProcessorSalesOrderBGaapFlow"*: the tasks are the same, although their result may differ.

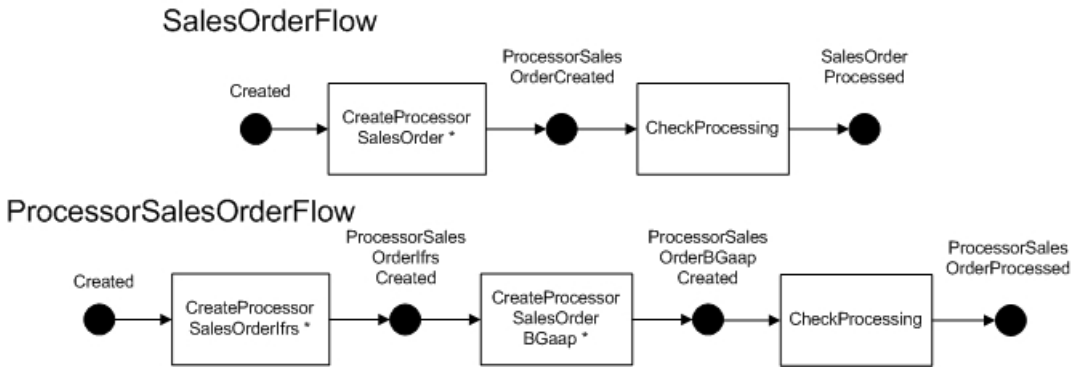

Figure 6.12: The SalesOrderFlow and its related processing flows

6.10.3.1 ProcessorSalesOrderIFRSFlow

We start with discussing the *"ProcessorSalesOrderIfrsFlow"*. The first task is *"CheckConcept"*, this task checks which concepts are relevant for SalesOrder by using the links between the data elements to trace the result stored in the data element *"ProcessorSaleIfrs*" related to to the SalesOrder. These concepts are *"Revenue"* and *"Trade Receivables"* and are stored in an attribute of the data element *"ProcessorSalesOrderIfrs"*.

The next task is the *"CreateConceptFlows"*, which creates the concept flows for each of the related concepts, being *"RevenueConceptFlow"* and *"TradeReceivablesConceptFlow"*. These flows execute the same tasks, although their results may differ. We graphically represent the RevenueConceptFlow in Figure 6.8. The first task in both flows is *"CheckRecognitionBusinessRules"*, this task checks which business rule is applicable, given the concepts and GAAP. In the example, for IFRS revenue is recognized at delivery date and trade receivables are recognized at invoice date. Since the current flow is the SalesOrderFlow, neither of the concepts should be recognized, therefore both the *"TradeReceivablesConceptFlow"* and the *"RevenueConceptFlow"* reach *"BRRecognitionNotFulfilled"* as a next status.

The next tasks in these flows now are *"CheckDisclosureBusinessRules"*, *"RevenueDisclosure"* and *"TradeReceivablesDisclosure"*, but these task do not have an implementation in the current version of the prototype. We return to the main flow: *"ProcessorSalesOrderIfrsFlow"* in which for IFRS no JournalEntries are created. The flow reaches its ending status *"JournalEntriesCompleted"*, without creating any JournalEntries.

6.10.3.2 ProcessorSalesOrderBGaapFlow

Now we discuss the *"ProcessorSalesOrderBGaapFlow"*. The first and second task are identical to the ProcessorSalesOrderIfrsFlow: the related concepts are traced and for each concept a flow is created, in the example these are the *"RevenueConceptFlow"* and the *"TradeReceivablesConceptFlow"*. The first task in both flows is *"CheckRecognitionBusinessRules"*, this task checks which business rule is applicable, given the concepts and GAAP. In the example, for Belgian Gaap revenue is recognized at order date and trade receivables are recognized at invoice date. Since the current flow is the SalesOrderFlow,

for the *"RevenueConceptFlow"* the status *"BRRecognitionFulfilled"* is reached, for the *"TradeReceivablesConceptFlow"* the status *"BRRecognitionNotFulfilled"* is reached.

The *"RevenueConceptFlow"* continues with the *"RevenueRecognition"* task. This task determines which accounts should be used, being *"Invoices to be prepared"* for debit and *"Revenue product X"* for credit and at which date the JournalEntry should be posted, being the date of the SalesOrder, 01/08/2016. The *"CheckMeasurementBusinessRules"* task checks which measurement business rules are applicable and finds one business rule that is applicable, being *"RevenueMeasurement"*. It is only the next task *"RevenueMeasurement"* that handles the content of this business rule: in this case the business rule states that the amount that should be recognized as revenue is the amount specified in the SalesOrder, being 20,000.

The next task is *"CheckPresentationBusinessRules"*, which does not have any result, so also the *"RevenuePresentation"* task does not have an implementation in this case. The same applies to *"CheckDisclosureBusinessRules"*, *"RevenueDisclosure"*, *"TradeReceivablesConceptFlow"*, and *"TradeReceivablesDisclosure"*.

Now we return to the *"ProcessorSalesOrderBGaapFlow"*: the one but last task *"CreateJournalEntry"* is a bridge task that creates a new JournalEntry. This JournalEntry has the date of the SalesOrder, 01/08/2016, a unique name and a reference to the Ledger in which the JournalEntry should be posted, being the Belgian GAAP Ledger. Then the *"JournalEntryFlow"* is triggered to create the needed JournalEntryLines.

The first task of the *"JournalEntryFlow"* (graphically represented in Figure 6.2) creates a JournalEntryLine, in the example: with the unique name JEL01, account *"Invoices to be prepared"*, debit, amount 20,000 and starting status *"created"*. This information comes from the *"ProcessorSalesOrderBGaapFlow"*. The next task checks whether other JournalEntryLines need to be created: in this case a second JournalEntryLine needs to be created so this task puts the status on *"JournalEntryNotCompleted"*, which triggers the *"CreateJournalEntryLine"* task again. Another JournalEntryLine is created: with the unique name JEL02, account *"Revenue product X"*, credit and amount 20,000. Again the *"CheckJournalEntryLines"* task is executed, which sets the ending status *"JournalEntryCompleted"*, since all JournalEntryLines for the JournalEntry are created.

The last task in the *"ProcessorSalesOrderBGaapFlow"* checks whether the *"JournalEntryFlow"* has reached its ending status, being *"JournalEntryCompleted"*, before the ending status of the *"ProcessorSalesOrderBGaapFlow"* is reached.

6.10.3.3 The Remainder of the SalesOrderFlow

Only when both the *"ProcessorSalesOrderBGaapFlow"* and the *"ProcessorSalesOrderIfrsFlow"* have reached an ending status, the ending status of the *"ProcessorSalesOrderFlow"* reaches its ending status *"ProcessorSalesOrderProcessed"* and only then the *"SalesOrderFlow"* reaches its ending status *"SalesOrderProcessed"*.

We show the result of the processing in the graphical user interface of the Prime Radiant in Figure 6.13: the SalesOrder is inserted into the graphical user interface, the Journal-

Entries and JournalEntryLines are created by the prototype as described above. Since all tasks are completed, the status of the SalesOrder is *"SalesOrderProcessed"*.

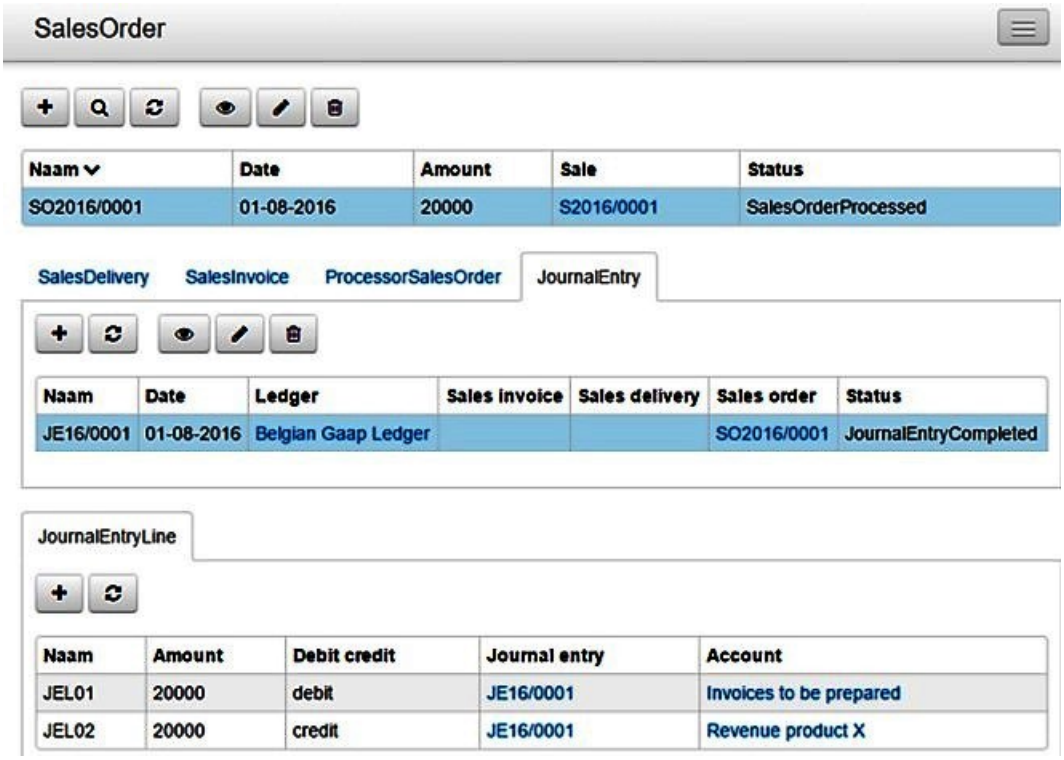

Figure 6.13: Screenshot in the Prime Radiant of a SalesOrder and its resulting JournalEntry and JournalEntryLines

6.10.4 The SalesDeliveryFlow

A new SalesDelivery needs to be created in the prototype by a user: providing a unique name, a date, a link to the belonging Sale, the belonging SalesOrder and a status (the first status we use is *"created"*). To automatically create one of these links, again the waterfall screens can be used, as described in Section 6.10.3. The status *"created"* triggers the *"SalesDeliveryFlow"* to be executed. Just like for Sale and SalesOrder, the same logic can be applied: the flow creates another data element *"ProcessorSalesDelivery"* and the second task checks whether the flow of that created data element reached its ending status. The *"ProcessorSalesDeliveryFlow"* is triggered and creates two other data elements, *"ProcessorSalesDeliveryIfrs"* and *"ProcessorSalesDeliveryBGaap"*, and the second task again checks whether the belonging flows of these data elements reached their ending status. These first two flows look exactly the same as the flows for SalesOrder in Figure 6.12. The *"ProcessorSalesDeliveryIfrsFlow"* and *"ProcessorSalesDeliveryBGaapFlow"* have the same tasks and states as the SalesInvoiceProcessorBGaapFlow in Figure 6.7.

Therefore, all tasks in *"ProcessorSalesDeliveryIfrsFlow"* and *"ProcessorSalesDeliveryBGaapFlow"* are also the same as the ones for SalesOrder. We discuss the differences in their execution in the following subsections.

6.10.4.1 ProcessorSalesDeliveryIfrsFlow

The first task checks which concepts are applicable, in the example Revenue and Trade Receivables. Therefore, both the *"RevenueConceptFlow"* and *"TradeReceivablesConceptFlow"* are initiated. When the recognition business rules are checked, the result for IFRS is that revenue should be recognized at delivery date so the resulting status for the *"RevenueConceptFlow"* is *"BRRecognitionFulfilled"* and trade receivables should be recognized at invoice date so the resulting status for *"TradeReceivablesConceptFlow"* is *"BRRecognitionNotFulfilled"*.

The *"RevenueConceptFlow"* continues with the *"RevenueRecognition"* task. This task determines which accounts should be used, being *"Invoices to be prepared"* for debit and *"Revenue product X"* for credit, and at which date the JournalEntry should be posted, being the date of the SalesDelivery, 03/08/2016. The *"CheckMeasurementBusinessRules"* task checks which measurement business rules are applicable and finds one business rule that is applicable, being *"RevenueMeasurement"*. It is only the next task *"RevenueMeasurement"* that handles the content of this business rule: in this case the business rule states that the amount that should be recognized as revenue is the amount specified in the SalesOrder. This amount should be traced back through the different relations between the data elements, until the belonging SalesOrder is found, which has an amount of 20,000.

The remainder of the tasks do not have an implementation: *"CheckPresentationBusinessRules"*, *"RevenuePresentation"*, *"CheckDisclosureBusinessRules"* and *"RevenueDisclosure"* for the *"RevenueConceptFlow"* and *"CheckDisclosureBusinessRules"* and *"TradeReceivablesDisclosure"* for the *"TradeReceivablesConceptFlow"*.

The one but last task of the *"ProcessorSalesDeliveryIfrsFlow"* then is *"CreateJournalEntry"* which is a bridge task that creates a new JournalEntry. This JournalEntry has the date of the SalesDelivery (03/08/2016), a unique name JE16/0002, a link to the SalesDelivery, a link to the SalesOrder, starting status *"created"* and is posted to the IFRS Ledger. Then the *"JournalEntryFlow"* is triggered to create the needed JournalEntryLines. The first JournalEntryLine is JEL03, the account *"Invoices to be prepared"* should be debited for an amount of 20,000. The second JournalEntryLine is JEL04, the account *"Revenue product X"* should be credited for an amount of 20,000.

The last task in the *"ProcessorSalesDeliveryIfrsFlow"* checks whether the *"JournalEntryFlow"* has reached its ending status, being *"JournalEntryCompleted"*, before the ending status of the *"ProcessorSalesDeliveryIfrsFlow"* is reached.

6.10.4.2 ProcessorSalesDeliveryBGaapFlow

The result of the CheckConcept task is the same as for IFRS: Revenue and Trade Receivables. Therefore, both the *"RevenueConceptFlow"* and *"TradeReceivablesConceptFlow"* are initiated. When the recognition business rules are checked the result for Belgian GAAP is that there are no business rules applicable for SalesDelivery, neither for revenue, nor for trade receivables. Both flows reach the status *"BRRecognitionNotFulfilled"*.

The remainder of the tasks (*"CheckDisclosureBusinessRules"*, *"RevenueDisclosure"* and *"TradeReceivablesDisclosure"*) do not have an implementation. Therefore, the *"ProcessorSalesDeliveryBGaapFlow"* ends without creating any JournalEntries.

6.10.4.3 The Remainder of the SalesDeliveryFlow

Only when both the *"ProcessorSalesDeliveryBGaapFlow"* and the *"ProcessorSalesDeliveryIfrsFlow"* have reached an ending status, the ending status of the *"ProcessorSalesDeliveryFlow"* reaches its ending status *"ProcessorSalesDeliveryProcessed"* and only then the *"SalesDeliveryFlow"* reaches its ending status *"SalesDeliveryProcessed"*.

We show the result of the processing in the graphical user interface of the Prime Radiant in Figure 6.14: the SalesDelivery is inserted into the graphical user interface, the JournalEntries and JournalEntryLines are created by the prototype as described above. Since all tasks are completed, the status of the SalesDelivery is *"SalesDeliveryProcessed"*.

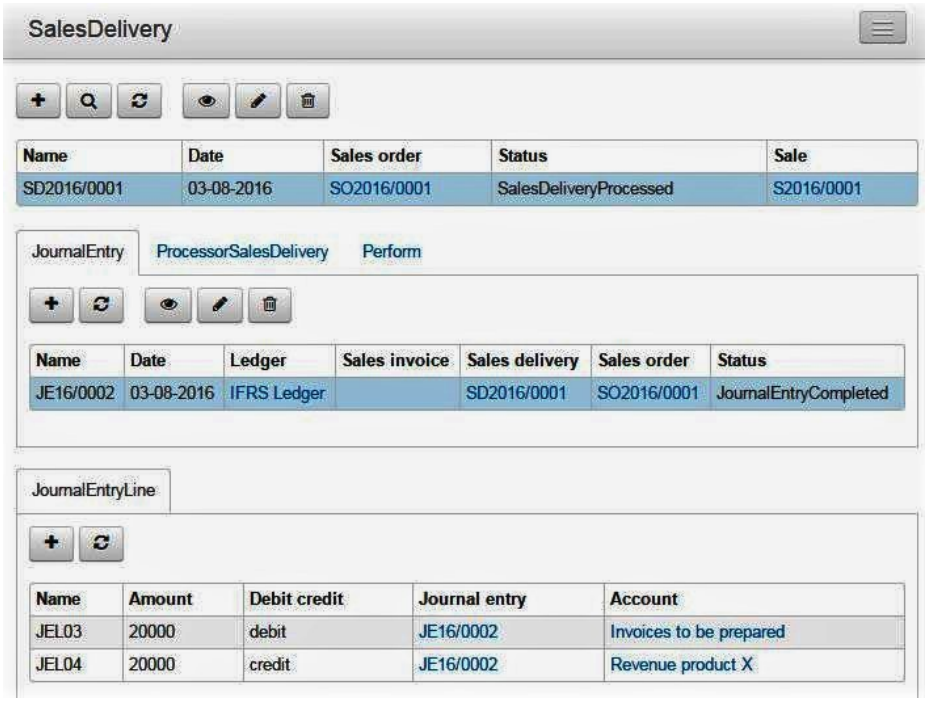

Figure 6.14: Screenshot in the Prime Radiant of a SalesDelivery and its resulting JournalEntry and JournalEntryLines

6.10.5 The SalesInvoiceFlow

A new SalesInvoice needs to be created in the prototype by a user: providing a unique name, a date, a link to the belonging Sale, the belonging SalesOrder and a status (the first status we use is *"created"*). To automatically create one of these links, again the waterfall screens can be used, as described in Section 6.10.3. The status triggers the execution of the *"SalesInvoiceFlow"*. The same logic as for Sale, SalesOrder and SalesDelivery can be applied: the flow creates another data element *"ProcessorSalesInvoice"*

and in the second task checks whether the flow of that created data element reached its ending status. The *"ProcessorSalesInvoiceFlow"* is triggered and creates two other data elements, *"ProcessorSalesInvoiceIfrs"* and *"ProcessorSalesInvoiceBGaap"*, and the second task again checks whether the flows of these data elements reached their ending status. These first two flows look exactly the same as the flows for SalesOrder in Figure 6.12. The *"ProcessorSalesInvoiceIfrsFlow"* and *"ProcessorSalesInvoiceBGaapFlow"* have the same tasks and states as the SalesInvoiceProcessorBGaapFlow in Figure 6.7. We discuss the differences in their execution in the following subsections.

6.10.5.1 ProcessorSalesInvoiceIfrsFlow

The first task checks which concepts are applicable, in the example Revenue and Trade Receivables. Therefore, both the *"RevenueConceptFlow"* and *"TradeReceivablesConceptFlow"* are initiated for IFRS. When the recognition business rules are checked, the result for IFRS is that revenue should be recognized at delivery date, so the resulting status for the *"RevenueConceptFlow"* is *"BRRecognitionNotFulfilled"* and trade receivables should be recognized at invoice date, so the resulting status for *"TradeReceivablesConceptFlow"* is *"BRRecognitionFulfilled"*.

The *"TradeReceivablesConceptFlow"* continues with the *"TradeReceivablesRecognition"* task. This task determines which accounts should be used, being *"Trade Receivables"* for debit and *"Invoices to be prepared"* for credit and at which date the JournalEntry should be posted, being the date of the SalesInvoice, 15/08/2016. The *"CheckMeasurementBusinessRules"* task checks which measurement business rules are applicable and finds one business rule that is applicable, being *"TradeReceivablesMeasurement"*. It is only the next task *"TradeReceivablesMeasurement"* that handles the content of this business rule: in this case the business rule states that the amount that should be recognized as Trade Receivables is the amount specified in the SalesInvoice, being 20,000.

The remainder of the tasks do not have an implementation: *"CheckPresentationBusinessRules"*, *"TradeReceivablesPresentation"*, *"CheckDisclosureBusinessRules"*, *"TradeReceivablesDisclosure"* and *"RevenueDisclosure"*.

Now we return to the *"ProcessorSalesInvoiceIfrsFlow"*: the one but last task is *"CreateJournalEntry"* which is a bridge task that creates a new JournalEntry. This JournalEntry has the date of the SalesInvoice (15/08/2016), a unique name JE16/0003, a link to the SalesInvoice, SalesDelivery and SalesOrder, starting status *"created"* and is posted to the IFRS Ledger. Then the *"JournalEntryFlow"* is triggered to create the needed JournalEntryLines. The first JournalEntryLine is JEL05, the account *"Trade Receivables"* should be debited for an amount of 20,000. The second JournalEntryLine is JEL06, the account *"Invoices to be prepared"* should be credited for an amount of 20,000.

The last task in the *"ProcessorSalesInvoiceIfrsFlow"* checks whether the *"JournalEntryFlow"* has reached its ending status, being *"JournalEntryCompleted"*, before the ending status of the *"ProcessorSalesInvoiceIfrsFlow"* is reached.

6.10.5.2 ProcessorSalesInvoiceBGaapFlow

The result of the CheckConcept task is the same as for IFRS, both the *"RevenueConceptFlow"* and *"TradeReceivablesConceptFlow"* are initiated. When the recognition business rules are checked the result for Belgian GAAP is that revenue should be recognized at delivery date, so the resulting status for the *"RevenueConceptFlow"* is *"BRRecognitionNotFulfilled"* and trade receivables should be recognized at invoice date, so the resulting status for *"TradeReceivablesConceptFlow"* is *"BRRecognitionFulfilled"*.

The *"TradeReceivablesConceptFlow"* continues with the *"TradeReceivablesRecognition"* task. This task determines which accounts should be used, being *"Trade Receivables"* for debit and *"Invoices to be prepared"* for credit and at which date the JournalEntry should be posted, being the date of the SalesInvoice, 15/08/2016. The *"CheckMeasurementBusinessRules"* task checks which measurement business rules are applicable and finds one business rule that is applicable, being *"TradeReceivablesMeasurement"*. It is only the next task *"TradeReceivablesMeasurement"* that handles the content of the business rule: in this case the business rule states that the amount that should be recognized as Trade Receivables is the amount specified in the SalesInvoice, being 20,000. The remainder of the tasks do not have an implementation: *"CheckPresentationBusinessRules"*, *"TradeReceivablesPresentation"*, *"CheckDisclosureBusinessRules"*, *"TradeReceivablesDisclosure"* and *"RevenueDisclosure"*.

Now we return to the *"ProcessorSalesInvoiceBGaapFlow"*: the one but last task is *"CreateJournalEntry"* which is a bridge task that creates a new JournalEntry. This JournalEntry has the date of the SalesInvoice (15/08/2016), a unique name JE16/0004, a link to the SalesInvoice, SalesDelivery and SalesOrder, the starting status *"created"* and is posted to the Belgian Gaap Ledger. Then the *"JournalEntryFlow"* is triggered to create the needed JournalEntryLines. The first JournalEntryLine is JEL07, the account *"Trade Receivables"* should be debited for an amount of 20,000. The second JournalEntryLine is JEL08, the account *"Invoices to be prepared"* should be credited for an amount of 20,000. The last task in the *"ProcessorSalesInvoiceBGaapFlow"* checks whether the *"JournalEntryFlow"* has reached its ending status, being *"JournalEntryCompleted"*, before the ending status of the *"ProcessorSalesInvoiceBGaapFlow"* is reached.

6.10.5.3 The Remainder of the SalesInvoiceFlow

Only when both the *"ProcessorSalesInvoiceBGaapFlow"* and the *"ProcessorSalesInvoiceIfrsFlow"* have reached an ending status, the ending status of the *"ProcessorSalesInvoiceFlow"* reaches its ending status *"ProcessorSalesInvoiceProcessed"* and only then the *"SalesInvoiceFlow"* reaches its ending status *"SalesInvoiceProcessed"*.

We show the result of the processing in the graphical user interface of the Prime Radiant in Figures 6.15 (for the IFRS details) and 6.16 (for the Belgian Gaap details): the SalesInvoice is inserted into the graphical user interface, the JournalEntries and JournalEntryLines are created by the prototype as described above. Since all tasks are completed, the status of the SalesInvoice is *"SalesInvoiceProcessed"*.

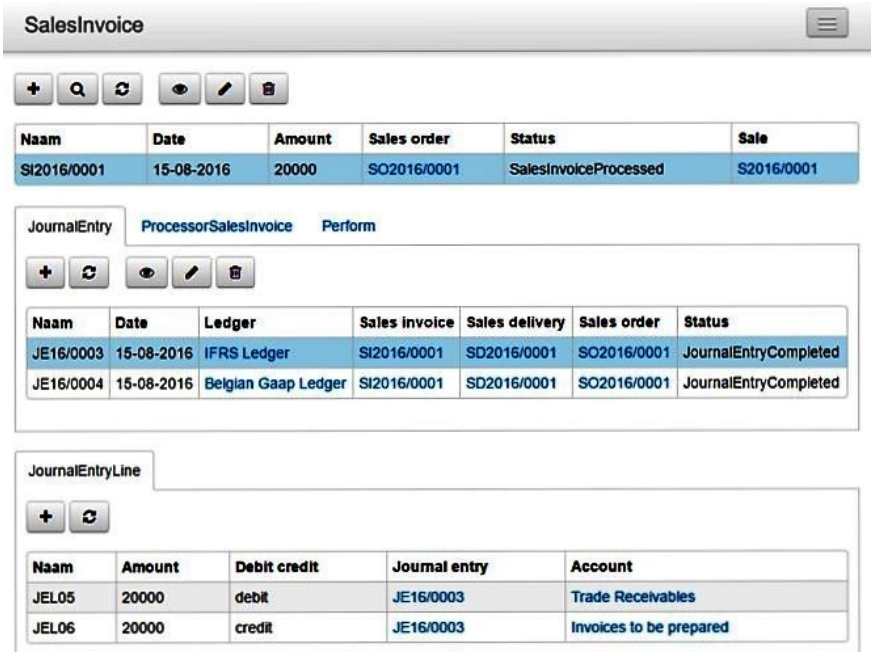

Figure 6.15: Screenshot in the Prime Radiant of a SalesInvoice and its resulting JournalEntry and JournalEntryLines: result for IFRS

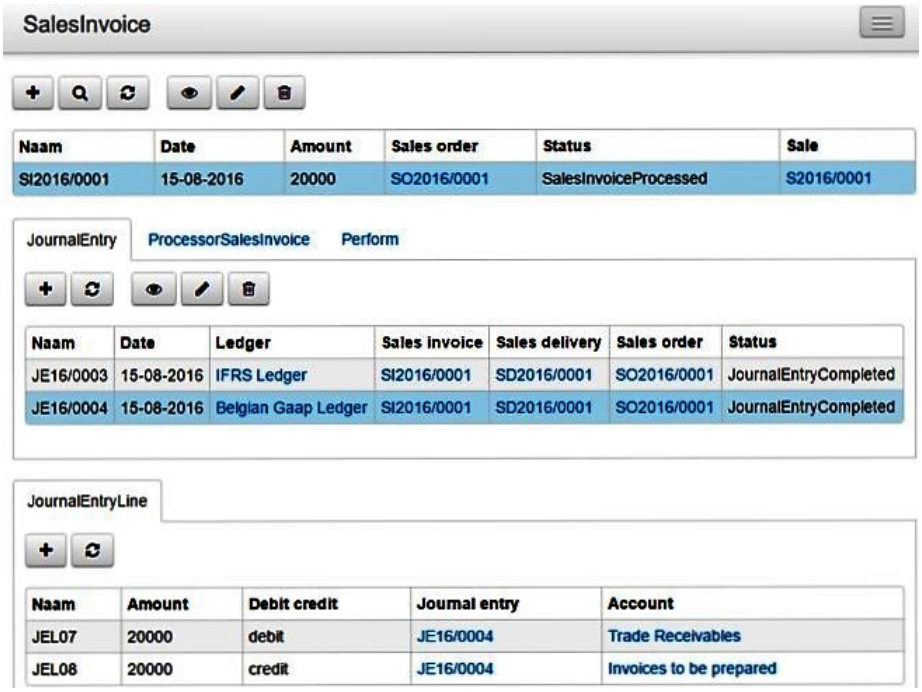

Figure 6.16: Screenshot in the Prime Radiant of a SalesInvoice and its resulting JournalEntry and JournalEntryLines: result for Belgian Gaap

6.11 Chapter Conclusions

In this chapter, we use the design principles of Chapter 5 to build a prototype AIS that supports processing of accounting-relevant events in multiple GAAP. With this prototype we contribute to literature by providing evidence that building an evolvable AIS (with respect to specific changes) is feasible. We provide practical evidence for the design principles of Chapter 5: they are valid and can be used. Moreover, it is feasible to build an evolvable (fine-grained) structure. The needed effort to build this structure is limited: the Prime Radiant generates a whole set of functionalities (like CRUD) after specifying the data elements, workflow elements, task elements and waterfall screens. Afterwards, only customizations need to be inserted and the generated software needs to be configured (pre-set of some data elements and relations and assembling the tasks in the workflows). We did not encounter any struggle or issue to build the evolvable structure of the prototype.

In this process, some additional insights are obtained regarding the limitations and required additions of these design principles in the context of building an AIS. The design principles have implications both at design time and at run time of the prototype: whereas some principles can be enforced during the definition of the model of the prototype, others (like using the same chart of accounts) have to be configured at run time. Therefore, additional documentation is needed for the configuration phase of the prototype, to aid users not to violate the design principles.

Chapter 7

Discussion, Conclusion, Limitations and Future Research

7.1 Discussion

In this dissertation we handle the problem domain of *"The design of evolvable AIS that support processing of accounting-relevant events in multiple GAAP"*. We focus on the issue of multiple GAAP in AIS and on the processing phase. We use a design science research approach, because we want to propose improvements for the design of multiple GAAP AIS regarding evolvability (March and Smith, 1995). The artifact we build and evaluate is a proof-of-concept prototype of an evolvable AIS that processes accounting-relevant events in multiple GAAP (Hevner et al., 2004). As methodological framework, we use the design science research process of Sonnenberg and vom Brocke (2012).

This process consists of four (core) activities: problem identification, design, construct and use (Sonnenberg and vom Brocke, 2012). After each activity, an evaluation activity is conducted (Sonnenberg and vom Brocke, 2012). Multiple authors (Hevner et al., 2004; March and Smith, 1995; Gregor and Hevner, 2013; Peffers et al., 2007) in design science research highlight the importance of evaluation throughout the design process, without suggesting a structured way to evaluate. However, the **F**ramework for **E**valuation in **D**esign **S**cience research (FEDS) (Venable et al., 2014) provides such a structured way for designing individual evaluation episodes in design science research. Therefore, we integrate the FEDS (Venable et al., 2014) into the design science research process of Sonnenberg and vom Brocke (2012).

We conduct the first three activities of the process of Sonnenberg and vom Brocke (2012) and their evaluation in this dissertation. In Figure 1.3, we showed how the different chapters in this dissertation relate to this process: Chapter 1 and 4 relate to *Identify Problem* and *Eval 1*, Chapter 5 relates to *Design* and *Eval 2*, and Chapter 6 relates to *Construct* and *Eval 3*. We discuss each activity of the process in more detail in the next sections. More details about the methodological approach are described in Chapter 3.

7.1.1 Discussion of Problem Identification Activity: Problem Statement and Detection of Evolvability Issues in Practice

7.1.1.1 Initial Problem Statement

Companies might need to adhere to multiple GAAP for a variety of reasons: compliance with local accounting and tax legislation, reporting to a parent company or other regulating mechanisms (for example, Solvency II). This requires a company to process its economic events according to multiple GAAP. We elaborate on this issue in Section 1.1 Moreover, these GAAP are not a static given, they change frequently. Different factors can influence the GAAP to which a company needs to comply: a change in regulatory environment (because of a merger or acquisition) or additional regulation for companies in a specific sector or with specific activities (for example, Solvency II for insurance companies). The GAAP themselves change by addressing new issues, elaborating guidance on specific issues or increasing reporting requirements to improve transparency, quality and relevance of financial information. We elaborate on this changing environment in Section 1.1.

Since companies rely heavily on IT to support their business, evolvability of information systems is crucial to be evolvable as an organization (Breu et al., 2002; Mannaert et al., 2012b; Huysmans, 2011). The accounting domain relies almost completely on information systems (Debreceny and Curtis, 2015), which makes it an interesting domain to study with regard to evolvability. Especially because accounting is also subject to change: regulatory demands that change over time, additional tasks for the accounting department (such as reporting of non-financial measures, audit of the information system,....) or changes in technology which drive innovation in the field (for example, XBRL reporting) (Grabski et al., 2011; American Accounting Association & American Institute of Certified Public Accountants, 2012). In Section 1.2 and 2.4, we identify evolvability as an important concern, especially with regard to information systems.

AIS literature suggests research should be done that relates to changes in the accounting domain that affect the information systems (Grabski et al., 2011), like research about XBRL (Pinsker and Li, 2008; Locke and Lowe, 2007), the Sarbanes-Oxley Act (Grabski et al., 2011) and IFRS reporting (Meall, 2004; Datardina et al., 2011; KPMG, 2008; Steele, 2009). However, we do not find relevant literature about evolvability in AIS. Moreover, little research is done about the multiple GAAP issue, since most research focuses on convergence to IFRS. Existing research identifies issues of compliance with multiple GAAP (Grabski et al., 2011; Fischer and Marsh, 2012; Meall, 2004), but does not propose any solutions to the problem. The REA model (Geerts et al., 2013; McCarthy, 1982; Geerts and McCarthy, 2002) is the most relevant literature we can find with regard to the research question. REA provides a shared and validated way of structuring accounting phenomena in the form of events, agents and resources (Geerts et al., 2013). We can situate this effort mainly in the *"recording"* phase of a global AIS environment (see Figure 1.2). When recording of accounting phenomena is done in compliance with REA, all needed conclusions and reporting can be derived from the primary data (McCarthy, 1982). In that respect, REA is interesting, since it provides evidence that when events are recorded according to the REA model, they can be processed in multiple GAAP.

However, REA does not provide us with detailed assistance in this matter: the question of how to process accounting phenomena to useful accounting information (in the form of for example, journal entries or an annual report) is not addressed in literature (Geerts and McCarthy, 2002). We elaborate on this research gap in Section 1.3. Therefore, this dissertation focuses on the multiple GAAP problem in AIS research and on the processing of accounting-relevant events in multiple GAAP.

As a solution to the issue of evolvability in information systems, we find Normalized Systems Theory that provides principles and design patterns for designing evolvable information systems (Mannaert et al., 2012b; Mannaert and Verelst, 2009). This approach is the most useful in literature, because it is grounded in systems theoretical stability and provides principles that are applicable in the development of software, whereas other approaches: 1) are empirical in nature and provide metrics that can only be used to study existing systems, 2) lack guidance for development of future systems because they leave room for interpretation, 3) are not unanimous in their principles, 4) lack guidance at higher levels of design (Mannaert et al., 2011, 2012b). Although Normalized Systems Theory is a software theory from origin, it has proven its usefulness for enterprise design (Huysmans, 2011) and business process design (Van Nuffel, 2011). Mannaert et al. (2012b) use the concept of combinatorial effects to define evolvability: an evolvable information system is an information system to which a set of anticipated changes can be applied without causing combinatorial effects. A combinatorial effect is caused when a change that is imposed on a system has an impact that is not only proportional to the change itself, but is also proportional to the size of the system on which the change is imposed (De Bruyn, 2014; Mannaert and Verelst, 2009).

In Section 1.4, we summarize the problem statement and lack of previous research as follows:

- The AIS literature does not study evolvability. We uncover references to evolvability in IT literature, where we find Normalized Systems Theory (which is mainly a software theory) as the most relevant, because it defines evolvability by using the concept of combinatorial effects;
- Multiple GAAP reporting is a burden for some companies, but we do not uncover guidance in literature on how companies can comply with multiple GAAP in an effective and efficient way by using their AIS;
- The changes in GAAP (both which GAAP and changes to the GAAP themselves) require considerable efforts and companies seem not to be able to deal with these changes in an evolvable way.

Summarizing this problem statement and research gap, we propose the following problem domain as subject for this dissertation: *The design of evolvable AIS that support processing of accounting-relevant events in multiple GAAP.*

7.1.1.2 Evaluating Initial Problem Statement

After identifying the problem, the first activity of the design science research process of Sonnenberg and vom Brocke (2012), we need to evaluate the activity. We summarize the

needed output for each phase in Table 3.1: for the first activity, we require a problem statement, a research need (research gap), design objectives, design theory and existing solutions to a practical problem. In the first evaluation activity, we need to evaluate whether the problem statement, research gap and design objectives can be justified.

In Chapter 1, we discuss the problem statement and research need extensively. We also put forward the design objective of evolvability and Normalized Systems Theory as design theory. However, we find little literature to support the problem statement and we do not consider existing practical solutions. Therefore, we turn to practice to study evolvability of multiple GAAP AIS by conducting case studies. In these case studies, we study practical solutions of multiple GAAP AIS and evaluate their evolvability. The research question we address is: *"Which violations of the evolvability criterion (combinatorial effects) are present in multiple GAAP AIS in practice?"*. These case studies strengthen the justification of the problem statement, research gap and design objectives. We discuss the rationale of using case studies in Section 4.1.

7.1.1.3 Case Studies of Multiple GAAP AIS

The use of case studies in a design science research process is not uncommon, but requires a mixed methods approach (Huysmans and De Bruyn, 2013). We elaborate on the mixed methods approach in Section 3.1. We use case studies during the problem identification phase of the design science research project, which makes the research design concurrent (first case studies then design science) exploratory (because the point of interface is the problem identification phase) (Huysmans and De Bruyn, 2013). The design science component is the main component, the case studies the supplementary component (Huysmans and De Bruyn, 2013).

We discuss the case study approach in Section 4.2: we use both case studies (five) and expert interviews (two). The case studies are exploratory in nature, because applying Normalized Systems Theory in the setting of AIS is new (Marshall and Rossman, 2006; Huysmans, 2011). We choose a collective case study approach and a heterogeneous sample (using different sectors) to increase generalizability of the results: identifying combinatorial effects that are inherent in the processing of accounting-relevant events in multiple GAAP in an AIS (Ritchie et al., 2003; Huysmans, 2011).

The number of case studies is limited, since we need a high level of detail to be able to analyze the results (Stake, 1994, 1998; Huysmans, 2011). We experience that the fifth case study only marginally adds to the results (we reach the point of theoretical saturation), giving us reasons to believe that this number of case studies suffices (Eisenhardt, 1989). Moreover this number falls within the range of four to ten cases, which is considered to be enough to draw valid conclusions (Eisenhardt, 1989).

We interview employees from the financial accounting department, as well as employees from the IT department involved in the AIS development. This provides us with more insight into the design of the AIS. The expert interviews are used to aid in analyzing the case studies and validating results.

In Section 4.4, we describe the designs of the AIS of the cases. First, we describe the modular structure (Figure 4.1) of the AIS from the cases (not adhering to multiple GAAP). The modular structure consists of the concepts: company code, event, entry processing module, entry processing tasks (concept, recognition, measurement, presentation and disclosure), journal entry, accounts, ledger and chart of accounts. Note that this modular structure considers events as already recorded and hence focuses on the processing of these events in accounting-relevant information. Second, we study in which way each case extends this modular structure to support multiple GAAP in their AIS. Two design choices are relevant in this respect: account design and posting design.

The account design provides a way to separate amounts, from different GAAP, posted to accounts. We identify three main account designs: duplicate accounts (account design 1), parallel ledgers (account design 2) and separate company codes (account design 3). Two cases use account design 1, the other three use account design 3. Although two of the cases which use account design 3, have custom-build software in which the set-up resembles account design 2.

The second design choice has to do with how journal entries are posted to the different accounts of the ledgers for the separate GAAP: by difference posting, by complete posting or by mixing difference posting and complete posting. Difference posting (posting design 1) means that all journal entries are posted to the accounts of the primary GAAP and only the difference between the primary and secondary GAAP is posted to the accounts of the secondary GAAP. The use of complete posting (posting design 2) means that separate journal entries are posted to the accounts of the different GAAP, even when the journal entries are identical. All cases in the sample use difference posting. Posting designs 3 and 4 mix the use of difference posting and complete posting: standard they use difference posting, but for a specific group of journal entries (for example financial instruments or fixed assets) they use complete posting. They do this, because for that group of journal entries, the accounting treatment is too different between the GAAP they use.

In certain combinations of account design and posting design, it is necessary to select the appropriate accounts to report in a specific GAAP. We identify two selection methods: selection method 1) determine which accounts to include and/or add and selection method 2) add all accounts, except for specific accounts not used in one of the GAAP.

7.1.1.4 Evaluating the AIS Designs with Respect to Evolvability

We evaluate the designs of Section 4.4 with respect to evolvability in Section 4.6. As we describe in Section 4.5, we use the FEDS (Venable et al., 2014) to increase the validity of the evaluation. In step one of the FEDS (Venable et al., 2014), we explicate the goals of evaluation: revealing combinatorial effects in AIS designs and providing evidence for the relevance of Normalized Systems Theory in the accounting domain. As for step two of the FEDS (Venable et al., 2014), we use a summative ex-ante artificial evaluation by proposing five (theoretical) changes to the AIS designs from the cases and evaluating whether the changes cause combinatorial effects in the designs. The evaluation is summative, because we do not intend to improve these designs in future iterations, rather we consider an AIS from practice and evaluate it. Later we use this evaluation to design

a prototype starting from scratch. Therefore, the evaluation is also considered ex-ante: before we actually construct an artifact (in our case the prototype). The evaluation is artificial, because we do not use a natural setting to evaluate the AIS designs.

In step three, we determine that the combinatorial effects are the property we want to evaluate. Combinatorial effects are a concept introduced by Normalized Systems Theory (Mannaert and Verelst, 2009; Mannaert et al., 2012b). Fourth, the individual evaluation episodes consist of a within-case analysis, followed by cross-case comparison (Benbasat et al., 1987; Eisenhardt, 1989). We report on these episodes in Section 4.6.

The five changes used in the evaluation are the following:

1. Creating a new account;
2. New version of an entry processing task for one GAAP (effect on journal entries);
3. New version of an entry processing task for one GAAP (effect on entry processing module);
4. New version of an entry processing task for all GAAP;
5. Adding a new GAAP to the AIS.

These changes reflect possible changes in/of GAAP that affect the processing of accounting-related events. The first four reflect changes in the GAAP, for example changing revenue recognition criteria. The last change reflects the change of the GAAP to which a company needs to comply, for example adding US GAAP as a GAAP to the system because of a merger.

To study the impact of each change, we illustrate different contexts. In that way we can structure evolvability problems. The first change, creating a new account, induces a combinatorial effect in account design 1 and 3a, but not in account designs 2, 3b and 3c.

The second change, a new version of an entry processing task for one GAAP, causes combinatorial effects in account designs 1b and possibly in 3b (depending on the specific implementation). However, the second change does not result in combinatorial effects for the other account designs. Then we consider the impact of a change in the entry processing module on the posting designs. We conclude that the second change induces combinatorial effects in posting designs 1, 3 and 4 (depending on which entry processing task changes), but not in posting design 2. Since posting designs 3 and 4 are mixtures of posting designs 1 and 2, we can conclude that using posting design 2 is the only way to prevent combinatorial effects that occur from new versions of an entry processing task.

Next, we consider the same change and study its impact on the entry processing module. The impact could depend on whether the entry processing tasks in the entry processing module are separated or not. However, we do not find a combinatorial effect resulting from this change.

As a fourth change, we consider that an entry processing task changes for all GAAP and we use the same contexts as for the third change: separation of entry processing tasks in the entry processing module or not. We conclude that this change causes a combinatorial effect in both contexts.

The last change of adding a new GAAP to the AIS only causes a combinatorial effect when account design 1b is used or selection method 1a is used. For all other designs the impact of the change is only proportional to the change.

Providing this overview of combinatorial effects, answers the proposed research question: we identify combinatorial effects present in multiple GAAP AIS from practice by using case studies. This is the first important contribution: we provide evidence that multiple GAAP AIS in practice contain combinatorial effects and therefore cannot be considered as evolvable AIS in the strict Normalized Systems Theory definition: an evolvable AIS following the Normalized Systems Theory definition is an AIS that does not contain combinatorial effects (Mannaert and Verelst, 2009; Mannaert et al., 2012b). Second, we show that Normalized Systems Theory, although a software theory from origin, can be used to evaluate AIS with regard to evolvability.

7.1.1.5 Contributions of the Problem Identification Activity

We contribute to the knowledge base (literature) in two ways. First, the combinatorial effects identified in Section 4.6 describe evolvability issues in multiple GAAP AIS from practice, their description is a contribution to the descriptive knowledge base (the knowledge base that provides descriptions of artifacts) (Gregor and Hevner, 2013). Second, the different implementations of AIS in practice (and relating them to literature) is a level 1 (situated artifact) contribution to the prescriptive knowledge base (Gregor and Hevner, 2013).

Next to the theoretical contributions, identifying the combinatorial effects is also relevant for practice. We provide evidence that the practical solutions for multiple GAAP AIS from the cases are not evolvable. Although there are differences in evolvability between the structures. This can aid practitioners in future AIS design choices. And what is more important, it reveals a theoretical foundation (the cause behind) for practical problems companies have in their multiple GAAP AIS. For example, when GAAP change, companies do not find it easy to change their AIS (Meall, 2004), which we can explain by the structure of the AIS: the structure of the AIS is not evolvable, which causes combinatorial effects when changes need to be made to the system.

Because of the exploratory nature of this dissertation, we need to acknowledge the limitations of this work. We choose to only study multiple GAAP AIS and limit ourselves to the processing of accounting-related events in AIS, this means we can only draw conclusions for this problem domain. Moreover, the case studies were limited to five and not all aspects of the AIS were studied in depth (e.g. the selection methods were not studied thoroughly), which limits the evolvability issues we find to those present in these cases, if other cases are studied additional issues might be revealed. Nevertheless we provide a contribution by raising the issue of evolvability in multiple GAAP AIS and providing practical evidence for the issue.

The case studies and their evaluation (the combinatorial effects) provide further justification for the initial problem statement: we found practical evidence that evolvability is an issue in multiple GAAP AIS, which is the problem we identify in the literature.

Therefore, we can argue that we have a justified problem statement (evolvability issues in multiple GAAP AIS) and research gap (no literature about evolvability in AIS and about design of multiple GAAP AIS) and design objectives (evolvability). This justification of the problem statement and the research gap is an important contribution to the relevance of the design science research process (also see Figure 1.3), justification of the design objectives, evolvability from Normalized Systems Theory (Mannaert and Verelst, 2009; Mannaert et al., 2012b), is important for the rigor of the process (Sonnenberg and vom Brocke, 2012; Hevner et al., 2004; Hevner, 2007).

7.1.2 Discussion of Design Activity: Deriving the Design Principles

The second activity in the design science research process (Sonnenberg and vom Brocke, 2012) is the design of an evolvable multiple GAAP AIS. Specifying the design of the artifact to build (the prototype) is the subject of this activity (Sonnenberg and vom Brocke, 2012). We develop design principles as the design specification (Sonnenberg and vom Brocke, 2012). Therefore, we propose the following research question: *Which design principles can be used to design evolvable AIS that can process accounting-related events according to multiple GAAP?* These design principles should guide future developers to design evolvable multiple GAAP AIS. Evaluation is key in the development of these design principles, since it provides validation for the design specification. Therefore, we use an iterative design in which we alternate development and evaluation. We use the FEDS (Venable et al., 2014) as a methodological framework for evaluation.

To develop the design principles, we study the combinatorial effects of Section 4.6 and their manifestations and conclude that some AIS design choices (described in Section 4.4) are more evolvable than others. This contrast constitutes the foundation to formulate the design principles. We structure the design principles, using the design choices that need to be made when designing a multiple GAAP AIS: account design, selection method, posting design and separation of entry processing tasks or not. After proposing a design principle, we evaluate it.

The goals of the evaluation (the first step of the FEDS (Venable et al., 2014)) are to check 1) if the design principles can be supported by current practice (the case studies) and existing literature and 2) to which extend the design principles prevent the combinatorial effects as described in Section 4.6. The second step in the FEDS (Venable et al., 2014) is to describe the why, when and how of the evaluation: we use a formative, ex-ante evaluation by providing a logical proof for the design principles. This logical proof (Hevner et al., 2004) relates the design principles to the case studies of Chapter 4 and relates them to relevant literature described in Chapter 2. Relating the design principles to the case studies improves relevance of the design principles (Hevner et al., 2004; Hevner, 2007), relating them to literature improves the rigor of the design principles (Hevner et al., 2004; Hevner, 2007). The logical proof is an artificial evaluation, since we do not use the design principles in a practical context. The third step in the FEDS (Venable et al., 2014) is to report the properties to evaluate: 1) whether companies in practice (the case studies we conduct) already use the design principles or not and 2) whether the design principles adhere to Normalized Systems Theory (Mannaert et al., 2012b; Mannaert and

Verelst, 2009; Van Nuffel, 2011; De Bruyn et al., 2012) and hence when implemented would prevent the combinatorial effects they are supposed to prevent. We report on the individual evaluation episodes (Venable et al., 2014) in Chapter 5.

These are the five resulting design principles, as an answer to the research question:

1. Journal entries for different GAAP should be posted in separate ledgers.
2. All GAAP should use the same chart of accounts.
3. Journal entries to different GAAP should be posted independently of each other.
4. Each event that can cause an accounting impact, should be processed by at least five separate tasks (versions of the entry processing tasks: concept, recognition criteria, measurement method, presentation, disclosure) before a journal entry can be posted.
5. Each entry processing task (concept, recognition criteria, measurement method, presentation, disclosure) that has a separate change driver should be separated in a distinct task, independent of the GAAP.

The design principles have implications on both the design of the artifacts of the modular system (design principle 2: the design of the chart of accounts) and on the design of the accounting business processes (design principles 1, 3, 4 and 5: the way journal entries are posted to a ledger). Therefore they can be considered as prescriptive knowledge and they are a level 2 contribution according to the framework of Gregor and Hevner (2013), being nacent design theory. The design principles will help future developers to design evolvable multiple GAAP AIS. Therefore, they contribute to both the knowledge base (rigor) as the environment (relevance) (Hevner, 2007).

Although these design principles provide necessary design principles that need to be followed when building evolvable multiple GAAP AIS, they are not sufficient to build evolvable multiple GAAP AIS. The reason for this is that the combinatorial effects we base ourselves on are not an exhaustive list of all combinatorial effects in multiple GAAP AIS: the case studies were limited in number and not all aspects of the multiple GAAP AIS were studied into detail, therefore, more combinatorial effects could be identified when studying more/different multiple GAAP AIS.

Next to this practical design contribution, the design principles contribute to Normalized Systems Theory literature by showing that with the use of the Normalized Systems Theory theorems (Mannaert and Verelst, 2009; Mannaert et al., 2012b), design principles can be developed that are applicable in the accounting domain. This can also be considered a contribution to the knowledge base (rigor) (Hevner, 2007).

The next step is to develop a prototype that adheres to the design principles. Therefore, we need additional design specifications: identifying data elements, their relationships and their attributes from the modular structure (see Figure 4.1). Next, we use the design principles of Chapter 5 to expand the functionality of the prototype: we design and describe how each design principle affects the design of the prototype. In every development step, we relate the design choices to Normalized Systems Theory (Mannaert and Verelst, 2009; Mannaert et al., 2012b). We report on this in Section 6.1.

7.1.3 Discussion of Construct Activity: Building the Prototype

The third activity of the design science research process is the construction phase (Sonnenberg and vom Brocke, 2012), where the design of the previous phase is used to construct an artifact. We specify the design of an evolvable multiple GAAP AIS by using the design principles from Chapter 5 to construct a prototype of an evolvable multiple GAAP AIS. Therefore, we propose the following research question: *What do we learn from implementing the design principles in a proof-of-concept AIS?* In Section 1.4, we refine this research question and we use the following sub-questions: Can we enforce the design principles? If not, how can we redefine them? What do we need in addition to the design principles to be able to build evolvable multiple GAAP AIS? How feasible is it to build evolvable multiple GAAP AIS? Therefore, we choose to build a prototype that has a specific, limited amount of functionality, but that provides actual examples. In this way, the design principles are actually translated into a tangible, concrete design and do not remain vague. This however results in the prototype not being able to be tested in a practical setting because of its limited functionality.

In Chapter 6, we describe how we iteratively build the prototype by using the Prime Radiant as a development tool. The use of a tool from Normalized Systems Theory literature (Mannaert and Verelst, 2009; Mannaert et al., 2012b) increases the rigor of the research (Hevner, 2007). After describing the design specification further in Section 6.1, we evaluate the first version of the prototype. Evaluation is an important step in the design science research process (Sonnenberg and vom Brocke, 2012). In this activity we also use the FEDS (Venable et al., 2014) as methodological framework. The first step in the framework is to set the goal of the evaluation: 1) does the prototype adhere to the design principles? 2) if it does, how? 3) are the design principles applicable in a prototype? 4) to which extend do they provide enough guidance? Second, we use a formative, ex-post evaluation by imposing changes (the same changes as in Section 4.6) to the prototype and evaluate whether they induce a combinatorial effect or not. This is the why, when and how of the evaluation, step two in FEDS (Venable et al., 2014). In the third step (Venable et al., 2014) we specify that the main property to evaluate is the evolvability of the prototype. The individual evaluation episodes, the fourth step (Venable et al., 2014), are described in Chapter 6, which are iterative in nature: we start with a prototype that only adheres to the first design principle, evaluate it, adjust it when necessary and then extend the prototype with the second design principle, evaluate it again and so on.

We conclude that the design principles have implications for the prototype both at design time and at run time. At design time the data elements, their relationships and attributes are defined, as shown in the ERD in Figure 6.1. But the first two design principles cannot be enforced at design time: at run time it is possible to create one ledger containing duplicated accounts (account design 1) and it is possible to use different charts of accounts for ledgers of other GAAP. We provide additional (additional to the design principles) documentation, so the prototype can be configured in accordance with the design principles. This documentation is more technical in nature: it explains how the design principles can be used when configuring a system, providing information about the data elements and their relationships. The configuration phase is the phase in between the implementation of the AIS and actually using the AIS for daily postings. Because of the level of detail of

the additional documentation, it also reveals the limitations of the design principles. For example, different companies can use different charts of accounts although they are part of the same AIS. In case of a need to create an additional account for each company (as the tax on labor example in Section 6.3), the change causes a combinatorial effect, since the account needs to be created in each company.

The third design principle requires a first (rudimentary) version of an entry processing module, which merely consists of Processor tasks executed after the occurrence of each identified event. Enforcing this third design principle can be done at design time. When we apply a change of an event processing task to the prototype at run time, the change does not induce a combinatorial effect.

Adding the fourth and fifth design principle requires adding more tasks and business rules to the prototype at design time. Then it becomes necessary to define workflows where tasks are reused, this can be done at run time, although we consider this as part of the configuration phase. Additionally, at run time it is necessary to keep the business rules up to date, following new versions of GAAP. Applying changes related to the entry processing tasks (new version for one GAAP or new version for all GAAP) to the last version of the prototype, does not reveal combinatorial effects. However, adding a new GAAP to the prototype does reveal combinatorial effects. These combinatorial effects are proportional to the number of events in the system and hence the number of processor flows in the system. We do not consider these combinatorial effects as evolvability issues because of two reasons: 1) the combinatorial effects are inherent in the change, being all events need to be processed in a specific way according to the new GAAP which needs to be determined in the system in some way and 2) most of the work that should be done when a new GAAP is added to the system can be automated by extending the functionality of the prototype, like for example creating the needed workflows automatically.

The main contribution to Normalized Systems Theory literature (the knowledge base, providing rigor for the research (Hevner, 2007)) of this third activity (constructing of a prototype) is showing the feasibility of building a prototype of an evolvable multiple GAAP AIS: we are able to build a prototype of an evolvable multiple GAAP AIS and because of using the Prime Radiant as a tool, a lot of functionality (e.g. CRUD) is generated automatically. Moreover, we show the practical relevance of the design principles of Chapter 5, a contribution to the environment, providing relevance for the research (Hevner, 2007). However, additional documentation is needed to assist users in the configuration phase of the prototype to enforce the first two design principles. As an additional contribution to the knowledge base (rigor), the prototype itself can be classified as a level 1 contribution and is prescriptive in nature (Gregor and Hevner, 2013).

7.1.4 Evolvability with Respect to a Set of Anticipated Changes

In this dissertation we contribute to designing multiple GAAP AIS by improving the evolvability of the design of the processing part of a multiple GAAP AIS. However, Normalized Systems Theory always considers evolvability with respect to a set of anticipated changes. This means that the evolvability of the prototype we build can only be claimed

as long as the set of changes are anticipated. The following changes are the ones we consider to be anticipated:

1. Creating a new account;
2. New version of an entry processing task for one GAAP (effect on journal entries);
3. New version of an entry processing task for one GAAP (effect on entry processing module);
4. New version of an entry processing task for all GAAP;
5. Adding a new GAAP to the AIS.

The first four changes reflect changes within a GAAP: we consider all possible changes in GAAP that can be expressed in journal entries. The last change is a change that reflects the case when a company needs to change the GAAP in which it needs to report, because of for example, additional regulation. Next to these changes, which we use as changes to test the evolvability of the prototype, we implicitly also test other changes which can be imposed on the prototype without adding combinatorial effects. This results in the following list of anticipated changes that can be applied to the prototype in an evolvable way:

- Add/update an event or subevent: we show in Section 6.5 that a new subevent can be added to the AIS without causing a combinatorial effect;
- Add/update an account: we consider this change in Section 6.3, an account can be added to the AIS without inducing a combinatorial effect;
- Add/update a journal entry: we show all through Chapter 6 that journal entries can be added to the prototype without causing a combinatorial effect;
- Add/update a journal entry line: we show all through Chapter 6 that journal entry lines can be added to the prototype without inducing a combinatorial effect;
- Add/update a GAAP: we consider this in Section 6.1 and 6.9, where we conclude that adding a GAAP does impose combinatorial effects, but those combinatorial effects are inherent in the change;
- Add/update an entry processing task (concept, recognition, measurement, presentation and disclosure): we consider this change in Sections 6.5, 6.7 and 6.8. Entry processing tasks can be updated without causing combinatorial effects;
- Add/update a ledger: we show in Section 6.9 that a new ledger can be added without causing a combinatorial effect.

If we compare this list with the modular structure (see Figure 4.1), all concepts mentioned in the modular structure are part of this list (and can thus be added or updated), except for Chart of Accounts and Company. The first one seems logical: design principle two indicates that the same chart of accounts should be used by all ledgers. Therefore, within the same company it is not relevant to add an additional chart of accounts. Adding or updating a company is another change that can be applied to the system: creating a new company, requires setting up new ledgers for all the GAAP the company uses. Events that occur in different companies should be separated, this can be done by adding a relation between the event and the company. The flows to process the events accounting-wise can

be duplicated for the new company. And if needed, new types of events can be created without combinatorial effects (as shown above). Hence, we can conclude that adding a new company does not cause combinatorial effects. That is why we can conclude that the prototype we build is evolvable with respect to all changes to the concepts identified in the modular structure of Figure 4.1.

It is important to note that this modular structure focuses on the processing of accounting-related events in multiple GAAP. Therefore, the contribution of this thesis and this list of anticipated changes is limited to the perspective of this processing activity and does not provide additional guidance to recording or reporting in AIS.

7.1.5 Concluding Remark Regarding the Problem Domain

The problem domain we address in this dissertation is the design of AIS that support processing of accounting-relevant events in multiple GAAP. Therefore, in the case studies we focus on this processing phase in the AIS of the case companies. The identified modular structure also focuses on this processing phase, since it considers events as given (recorded). Since the rest of the dissertation uses this modular structure as starting point, the focus remains on the processing of the accounting-relevant events: the changes used to identify the combinatorial effects are based on changes in/of GAAP that affect the processing of the events in multiple GAAP. Hence, the combinatorial effects are effects that occur in this processing phase. The design principles are derived from these combinatorial effects and the prototype is based on the design principles and therefore the processing phase remains the focus throughout this dissertation.

7.2 Conclusion

In this dissertation we use a mixed methods approach (Huysmans and De Bruyn, 2013) of design science and case studies. We adhere to the design science research process of Sonnenberg and vom Brocke (2012) and we use the FEDS framework Venable et al. (2014) for evaluation purposes after each activity. This dissertation uses this approach to contribute to a solution for the problem of evolvable multiple GAAP AIS.

In Chapter 1, we identify the problem at hand: evolvability in multiple GAAP AIS. We discuss evolvability as an important concern in Information Systems and show its relevance in the field of AIS. In software design Normalized Systems Theory provides guidance to design evolvable information systems by proposing four theorems. We show there is a lack of literature about evolvability in AIS and about multiple GAAP, but we do find a call for research about multiple GAAP in the AIS literature: REA literature focuses on the recording of events so they can be processed in multiple GAAP, but lacks the focus on this processing phase. Therefore, we address the issue of processing accounting-related events in multiple GAAP. Although we can justify the problem of evolvable multiple GAAP AIS, we can only provide a justification based on literature and we are not sure whether a practical solution for the problem exists. Therefore, this research is exploratory in nature and we conduct case studies to study multiple GAAP AIS in practice.

Chapter 4 reports on these case studies. We describe existing AIS designs from five case companies, resulting in a general modular structure for the AIS and different design choices that are made to extend this structure to support the processing of events in multiple GAAP. In this exploratory setting, we find this number of case studies sufficient to be able to draw some conclusions with regard to evolvability in multiple GAAP AIS in practice. We use Normalized Systems Theory to evaluate the AIS designs from the case studies with regard to evolvability. The result is the identification of violations of evolvability in accordance with Normalized Systems Theory, which we call combinatorial effects (Mannaert et al., 2012b; Mannaert and Verelst, 2009). These violations are related to the processing of accounting-relevant events in multiple GAAP.

In Chapter 5, we use these combinatorial effects to develop design principles. These design principles provide guidance for the design of future evolvable multiple GAAP AIS. We evaluate them theoretically by relating them to the case studies and to the Normalized Systems Theory literature. The result is the following set of five design principles:

1. Journal entries for different GAAP should be posted in separate ledgers.
2. All GAAP should use the same chart of accounts.
3. Journal entries to different GAAP should be posted independently of each other.
4. Each event that can cause an accounting impact, should be processed by at least five separate tasks (versions of the entry processing tasks: concept, recognition criteria, measurement method, presentation, disclosure) before a journal entry can be posted.
5. Each entry processing task (concept, recognition criteria, measurement method, presentation, disclosure) that has a separate change driver should be separated in

a distinct task, independent of the GAAP.

In Chapter 6, a prototype is developed that serves as a proof-of-concept for the design principles of Chapter 5. This means that we found it more important to build a prototype that is evolvable and demonstrates the design principles with some examples, than building a prototype that contains more functionality but does not show the relevance of the design principles. Therefore, the final prototype does not contain enough functionality to be tested in practice. We find that building this prototype is feasible: we provide evidence that the design principles can be used to build evolvable multiple GAAP AIS, which is facilitated by the Prime Radiant tool that generates a large part of the software itself so only configuration and customizations need to be inserted. However, to enforce some of the design principles, we need additional documentation to assist the user in configuring the prototype.

This dissertation contributes to the AIS literature and practice in several ways. In Chapter 4, we provide evidence that multiple GAAP AIS in practice contain combinatorial effects and therefore cannot be considered as evolvable AIS in the strict Normalized Systems Theory definition (Mannaert and Verelst, 2009; Mannaert et al., 2012b). This can aid practitioners in future AIS design choices and what is more important, it reveals theoretical foundation for practical problems companies have when they process accounting-relevant events in multiple GAAP (being able to identify the cause behind practical problems). In the classification of Gregor and Hevner (2013), the combinatorial effects identified from the case studies contribute to the descriptive knowledge base. They are the first of their kind in AIS literature and therefore provide possibilities for follow-up research that extends the list of combinatorial effects, both with more in depth analysis of multiple GAAP AIS as with combinatorial effects in other AIS problem domains.

Situating different implementations of AIS in practice (and relating them to literature) is a level 1 contribution to the prescriptive knowledge base (Gregor and Hevner, 2013). The identification of the combinatorial effects is based on the modular structure of multiple GAAP AIS which focuses on the processing of accounting-related events in multiple GAAP and does not consider how to record these events. This is the problem domain we want to address in this dissertation: recording of events is extensively researched in REA literature, but literature lacks details about the processing of these events.

The design principles of Chapter 5 have implications for both the design of the artifacts of the modular system (design principle 2: the design of the chart of accounts) as for the design of accounting business processes (design principles 1, 3, 4 and 5: the way postings are made to ledgers). Therefore they can be considered as prescriptive knowledge and they are a level 2 contribution according to the framework of Gregor and Hevner (2013), being nacent design theory. Moreover, the design principles will help future designers to develop evolvable multiple GAAP AIS: they provide necessary conditions to develop the processing part of evolvable multiple GAAP AIS, however these principles are not sufficient and additional research is needed to extend design principles in this area.

The construction of the prototype in Chapter 6 shows the feasibility of building a prototype of an evolvable multiple GAAP AIS that processes accounting-relevant events in multiple GAAP. Moreover, it shows the practical relevance of the design principles of Chapter 5. As a contribution to the knowledge base, the prototype can be classified

as a level 1 contribution and is prescriptive in nature (Gregor and Hevner, 2013). This prototype is limited in functionality and only serves as a proof-of-concept, nevertheless it can evolve (in the Normalized Systems Theory way: keep on growing in functionality and complexity, without causing combinatorial effects) by extending it with more functionality over time.

Lastly, we want to discuss our contribution to Normalized Systems Theory: we show throughout this entire dissertation that Normalized Systems Theory can be applied in the field of accounting. We use Normalized Systems Theory to evaluate existing AIS design with regard to evolvability. The design principles are developed by using the Normalized Systems Theory theorems and applying them in the accounting domain. We demonstrate that building a prototype of an AIS that adheres to the Normalized Systems Theory theorems is feasible, which proves the practical relevance of Normalized Systems Theory in the accounting domain.

7.3 Limitations

This dissertation is exploratory in nature: we study a problem not addressed before in literature. Therefore, there are several limitations to this work that are inherent to exploratory research (Marshall and Rossman, 2006).

First, we choose a specific AIS issue to study, being the processing of accounting-related events in multiple GAAP. However, other issues might also be relevant: for example, internal reporting of financial information to management in the context of business intelligence or cost accounting.

Second, we limit ourselves to studying the processing of accounting-relevant events, not the recording or reporting of these events. Since the REA ontology is a mature domain in the recording phase of the accounting process, it might be interesting to study evolvability of the REA model. We also do not consider disclosures that cannot be expressed in journal entries, but require additional reporting of financial information in the notes to the financial statement.

Third, the number of cases is limited to five. Moreover, not all aspects of the AIS are studied into depth. For example, regarding selection methods we did not conduct thorough analyses. Therefore, we cannot be sure that we identify a general modular structure applicable to all AIS and we cannot be sure that we uncover all possible design choices to extend this modular structure to support the processing of events in multiple GAAP.

Fourth, when searching for evolvability issues, we only propose five changes to the system, which limits the identification of combinatorial effects and hence also limits the resulting design principles and functionality of the prototype. Therefore, the design principles can be considered as necessary but not sufficient in building evolvable multiple GAAP AIS. However, we did show in Section 7.1.4 that with respect to the identified modular structure, we consider all possible changes.

Fifth, the prototype has limited functionality: we only implement some examples regarding revenue, this cannot be considered a comprehensive AIS testable by companies in practice. Therefore, this prototype should first be extended, before an actual fourth *"use"* activity (the fourth activity in the design science research process of Sonnenberg and vom Brocke (2012)) can be conducted and evaluated.

7.4 Further Research

Despite its limitations, we believe that the exploratory nature of this dissertation, is an important first step for further research. Both because the multiple GAAP issue can be studied into more depth and because it might convince other parties of the feasibility of building evolvable AIS and therefore be a starting point to study similar evolvability issues.

The issue of multiple GAAP can be studied into more depth by conducting additional case studies (extending the width of the research) and going into more detail in these case studies (extending the depth of the research) to be able to refine the modular structure of Figure 4.1 and reveal additional design choices. In that way additional combinatorial effects can be revealed, the design principles can be elaborated and the functionality of the prototype can be extended. For example, we did not address presentation and disclosure requirements of GAAP that require another format than a journal entry. Conducting multiple iterations of this process and analyzing additional examples of postings, can then lead to a prototype that is fit for testing in a real-live setting, so the fourth activity of the design science research process of Sonnenberg and vom Brocke (2012) *"use"* can be executed and evaluated.

However, before such a prototype can be developed, additional issues in AIS design should be researched. For example, evolvability of recording of events: REA provides guidance to record events in AIS, but the REA model has only been studied by Vanhoof (2010) with regard to evolvability. Vanhoof (2010) reveals that the REA ontology as such is not evolvable and proposes solutions in concrete cases, but these results cannot be generalized. Other issues that can be studied are for example, the definition of selection criteria for reporting (as mentioned in Section 5.2), reporting in XBRL and calculating cash flows by using the direct method.

Next to the financial account domain (being external reporting), evolvability in AIS can be extended by researching issues in management accounting, cost accounting, business intelligence and audit. Also, research opportunities are present in other business domains like logistics, production etc. Moreover, additional research efforts to develop additional principles at the business level (like Van Nuffel (2011)) and the enterprise level (like Huysmans (2011); De Bruyn (2014)) can provide additional structure to the design process and aid future designers in all domains to design more evolvable information systems.

Appendices

Appendix A

Explanation of Entity-relationship Diagram Notation

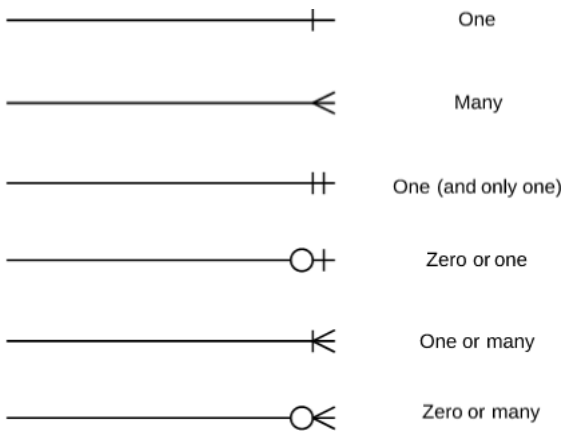

Figure A.1: Explanation of Entity-relationship diagram notation

Bibliography

Abran, M. and Moore, J., editors. *Guide to the Software Engineering Body of Knwoedge (SWEBOK)*. IEEE Computer Society, 2004.

Allen, B. R. and Boynton, A. C. Information Architecture: In Search of Efficient Flexibility. *MIS Quarterly*, 15(4):435–445, 1991.

Alles, M., Brennan, G., Kogan, A., and Vasarhelyi, M. A. Continuous Monitoring of Business Process Controls: A Pilot Implementation of a Continuous Auditing System at Siemens. *International Journal of Accounting Information Systems*, 7(2):137 – 161, 2006. 2005 Research Symposium on Integrity, Privacy, Security & Trust in an IT Context.

Alles, M. G., Kogan, A., and Vasarhelyi, M. A. Putting Continuous Auditing Theory into Practice: Lessons from Two Pilot Implementations. *Journal of Information Systems*, 22(2):195–214, 2008.

American Accounting Association & American Institute of Certified Public Accountants. The Pathways CommissionCharting a National Strategy for the Next Generation of Accountants. Technical report, AAA/AICPA, Sarasota, Florida, July 2012.

Anda, B. Assessment of Software System Evolvability. In *Ninth International Workshop on Principles of Software Evolution: In Conjunction with the 6th ESEC/FSE Joint Meeting*, IWPSE '07, pages 71–74, New York, NY, USA, 2007. ACM.

Arnheiter, E. D. and Harren, H. A Typology to Unleash the Potential of Modularity. *Journal of Manufacturing Technology Management*, 16(7):699–711, 2005.

Arsanjani, A., Booch, G., Boubez, T., Brown, P. C., Chappell, D., deVadoss, J., Erl, T., Josuttis, N., Krafzig, D., Little, M., Loesgen, B., Thomas Manes, A., McKendrick, J., Ross-Talbot, S., Tilkov, S., Utschig-Utschig, C., and Wilhelmsen, H. SOA Manifesto. Technical report, SOA Manifesto Working Group, 2009.

Bajgoric, N. Web-based Information Access for Agile Management. *International Journal of Agile Management Systems*, 2(2):121–129, 2000.

Baldwin, C. and Clark, K. Managing in an Age of Modularity. *Harvard Business Review*, 75(5):84 – 93, 1997.

Baldwin, C. and Clark, K. *Design Rules, Volume 1: The Power of Modularity*. MIT Press Books. The MIT Press, 2000.

Beer, M. and Nohria, N. Cracking the Code of Change. *Harvard Business Review*, 78(3): 13–23, 2000.

Benbasat, I. and Zmud, R. W. Empirical Research in Information Systems: the Practice of Relevance. *MIS Quarterly*, 23(1):3 – 16, 1999.

Benbasat, I., Goldstein, D. K., and Mead, M. The Case Research Strategy in Studies of Information Systems. *MIS Quarterly*, 11(3):369 – 386, 1987.

Bernardes, E. S. and Hanna, M. D. A Theoretical Review of Flexibility, Agility and Responsiveness in the Operations Management Literature. *International Journal of Operations & Production Management*, 29(1/2):30 – 53, 2009.

Boltzmann, L. *Lectures on Gas Theory*. Dover Publications, 1995.

Börjesson, A. and Mathiassen, L. Improving Software Organizations: Agility Challenges and Implications. *Information Technology & People*, 18(4):359–382, 2005.

Botha, A., Kourie, D., and Snyman, R. *Coping with Continuous Change in the Business Environment: Knowledge Management and Knowledge Management Technology*. Chandos Publishing, 2008.

Breu, K., Hemingway, C. J., Strathern, M., and Bridger, D. Workforce Agility: the New Employee Strategy for the Knowledge Economy. *Journal of Information Technology (Routledge, Ltd.)*, 17(1):21 – 31, 2002.

Campagnolo, D. and Camuffo, A. The Concept of Modularity in Management Studies: A Literature Review. *International Journal of Management Reviews*, 12(3):259 – 283, 2010.

Chang, J. C.-J. and King, W. R. Measuring the Performance of Information Systems: A Functional Scorecard. *J. Manage. Inf. Syst.*, 22(1):85–115, January 2003.

Chung, S. H., Byrd, T. A., Lewis, B. R., and Ford, F. N. An Empirical Study of the Relationships between IT Infrastructure Flexibility, Mass Customization, and Business Performance. *SIGMIS Database*, 36(3):26–44, August 2005.

Ciraci, S. and van den Broek, P. Evolvability as a Quality Attribute of Software Architectures. In Duchien, L., D'Hondt, M., and Mens, T., editors, *Proceedings of the International ERCIM Workshop on Software Evolution 2006*, pages 29–31, Mons, April 2006. UMH.

Columbus, L. Gartner's ERP Market Share Update Shows The Future Of Cloud ERP Is Now. *Forbes*, 2014.

Computer Profile. SAP most common ERP application in Big Business. Electronically Published: http://www.computerprofile.com/analytics-papers/sap-most-common-erp-application-in-big-business/, March 2016.

Conboy, K. and Fitzgerald, B. Toward a Conceptual Framework of Agile Methods: A Study of Agility in Different Disciplines. In *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research*, WISER '04, pages 37–44, New York, NY, USA, 2004. ACM.

Cook, S., Ji, H., and Harrison, R. Software Evolution and Evolvability. Technical report, University of Reading, 2000.

Cook, S., Harrison, R., and Wernick, P. Information System Evolvability, Feedback and Pattern Languages. In *IEE Proceedings-Software*, volume 153 of *IEE Proceedings*, pages 137–148. Institution of Engineering and Technology, IEEE Computer Society, August 2006.

Datardina, M., Wan, M., and Chiu, K. IFRS Implications of IT ACC 626: I.T. Assurance & Computer-Assisted Auditing Technique Section 2. Electronically Published, july 2011.

De Bruyn, P. *Generalizing Normalized Systems Theory : Towards a Foundational Theory for Enterprise Engineering*. Ph.d. dissertation of the faculty of applied economic sciences - department of management information systems, University of Antwerp, Antwerp, November 2014.

De Bruyn, P., Van Nuffel, D., Huysmans, P., and Mannaert, H. Towards Functional and Constructional Perspectives on Business Process Patterns. In *Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA)*, pages 459–464, Barcelona,, 2011. Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011).

De Bruyn, P., Nuffel, D., Verelst, J., and Mannaert, H. Towards Applying Normalized Systems Theory Implications to Enterprise Process Reference Models. In Albani, A., Aveiro, D., and Barjis, J., editors, *Advances in Enterprise Engineering VI*, volume 110 of *Lecture Notes in Business Information Processing*, pages 31–45. Springer Berlin Heidelberg, 2012.

De Bruyn, P., Huysmans, P., Mannaert, H., and Verelst, J. Understanding Entropy Generation During the Execution of Business Process Instantiations: An Illustration from Cost Accounting. In Proper, H. A., Aveiro, D., and Gaaloul, K., editors, *Advances in Enterprise Engineering VII*, volume 146 of *Lecture notes in business information processing*, pages 103–117. Springer International Publishing, 2013.

De Bruyn, P., Van Nuffel, D., Huysmans, P., and Mannaert, H. Confirming Design Guidelines for Evolvable Business Processes based on the Concept of Entropy. *International journal on advances in software*, 7(1 & 2):341–352, 2014.

de Vasconcelos, J. B., Kimble, C., Carreteiro, P., and Rocha, l. The Application of Knowledge Management to Ssoftware Evolution. *International Journal of Information Management*, pages –, 2016.

Debreceny, R. S. Betwixt and Between? Bringing Information Systems and Accounting Systems Research Together. *Journal of Information Systems*, 25(2):1 – 9, 2011.

Debreceny, R. S. and Curtis, M. B. Challenges From and To the Senior Editors of the Journal of Information Systems. *Journal of Information Systems*, 29(1):1–8, 2015.

Ding, Y., Hope, O.-K., Jeanjean, T., and Stolowy, H. Differences between domestic accounting standards and IAS: Measurement, determinants and implications. *Journal of Accounting and Public Policy*, 26(1):1 – 38, 2007.

Djelic, M.-L. and Ainamo, A. The Coevolution of New Organizational Forms in the Fashion Industry: A Historical and Comparative Study of France, Italy, and the United States. *Organization Science*, 10(5):622 – 637, 1999.

Dove, R. *Response Ability: the Language, Structure, and Culture of the Agile Enterprise*. John Wiley & Sons, 2002.

Dubé, L. and Paré, G. Rigor in Information Systems Positivist Case Research: Current Practices, Trends, and Recommendations. *MIS Quarterly*, 27(4):597 – 635, 2003.

Dunn, C. L. and McCarthy, W. E. The REA Accounting Model: Intellectual Heritage and Prospects for Progress. *Journal of Information Systems*, 11(1):31 – 51, 1997.

Eisenhardt, K. M. Building Theories from Case Study Research. *The Academy of Management Review*, 14(4):pp. 532–550, 1989.

Ernst, D. Limits to modularity: reflections on recent developments in chip design. *Industry and Innovation*, 12(3):303–335, 2005.

Ethiraj, S. K. and Levinthal, D. Modularity and Innovation in Complex Systems. *Management Science*, 50(2):159–173, 2004.

Ethiraj, S. K., Levinthal, D., and Roy, R. R. The dual role of modularity: innovation and imitation. *Management Science*, 54(5):939–955, 2008.

Ferguson, C. and Seow, P.-S. Accounting Information Systems Research over the Past Decade: Past and Future Trends. *Accounting & Finance*, 51(1):235 – 251, 2011.

Fink, L. and Neumann, S. Gaining Agility through IT Personnel Capabilities: The Mediating Role of IT Infrastructure Capabilities. *Journal of the Association for Information Systems*, 8(8):440 – 462, 2007.

Fischer, M. and Marsh, T. Accounting and Reporting Convergence. *International Journal of the Academic Business World*, 6(1):1 – 10, 2012.

Fisher, I. E. A Prototype System for Temporal Reconstruction of Financial Accounting Standards. *International Journal of Accounting Information Systems*, 8(3):139–164, 2007.

Garud, R. and Kumaraswamy, A. Technological and Organizational Designs for Realizing Economies of Substitution. *Strategic management journal*, 16(S1):93–109, 1995.

Gebauer, J. and Schober, F. Information System Flexibility and the Cost Efficiency of Business Processes. *Journal of the Association for Information Systems*, 7(3):122 – 146, 2006.

Geerts, G. L. *Towards a New Paradign in Structuring and Processing Accounting Data*. Ph.d. dissertation, Free University of Brussels, Brussels, March 1993.

Geerts, G. L. A Design Science Research Methodology and its Application to Accounting Information Systems Research. *International Journal of Accounting Information Systems*, 12(2):142 – 151, 2011. Special Issue on Methodologies in AIS Research.

Geerts, G. L. and McCarthy, W. E. Modelling Business Enterprises as Value–Added Process Hierarchies with Resource–Event–Agent Object Templates. In Sutherland, J., Patel, D., Casanave, C., Hollowell, G., and Miller, J., editors, *Business Object Design and Implementation*, pages 94–113. Springer - verlag, 1997.

Geerts, G. L. and McCarthy, W. E. An Accounting Object Infrastructure for Knowledge-Based Enterprise Models. *IEEE Expert Intelligent Systems & Their Applications*, 14 (3):89, 1999.

Geerts, G. L. and McCarthy, W. E. The Ontological Foundation of REA Enterprise Information Systems. In *American Accounting Association Conference*, august 2000a.

Geerts, G. L. and McCarthy, W. E. Augmented Intensional Reasoning in Knowledge–Based Accounting Systems. *Journal of Information Systems*, 14(2):127, 2000b.

Geerts, G. L. and McCarthy, W. E. Using Object Templates from the REA Accounting Model to Engineer Business Processes and Tasks. *The Review of Business Information Systems*, 5(4), april 2001.

Geerts, G. L. and McCarthy, W. E. An Ontological Analysis of the Economic Primitives of the Extended – REA Enterprise Information Architecture. *International Journal of Accounting Information Systems*, 3(1):1, 2002.

Geerts, G. L., Graham, L. E., Mauldin, E. G., McCarthy, W. E., and Richardson, V. J. Integrating Information Technology into Accounting Research and Practice. *Accounting Horizons*, 27(4):815 – 840, 2013.

Gershenson, J. K., Prasad, G. J., and Zhang, Y. Product Modularity: Definitions and Benefits. *Journal of Engineering Design*, 14(3):295–313, 2003.

Giachetti, R. E., Martinez, L. D., Senz, O. A., and Chen, C.-S. Analysis of the Structural Measures of Flexibility and Agility Using a Measurement Theoretical Framework. *International Journal of Production Economics*, 86(1):47, 2003.

Grabski, S. V., Leech, S. A., and Schmidt, P. J. A Review of ERP Research: A Future Agenda for Accounting Information Systems. *Journal of Information Systems*, 25(1): 37 – 78, 2011.

Gregor, S. and Hevner, A. R. Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, 37(2):337, 2013.

Gregor, S. and Jones, D. The Anatomy of a Design Theory. *Journal of the Association for Information Systems*, 8(5):313 – 335, 2007.

Gruber, T. R. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5:199–220, 1993.

Guan, J., Levitan, A. S., and Kuhn Jr., J. R. How AIS can Progress Along with Ontology Research in IS. *International Journal of Accounting Information Systems*, 14(1):21 – 38, 2013.

Hammer, M. and Champy, J. *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Business, 1993.

Harmon, P. *Business Process Change: A Business Process Management Guide For Managers and Process Professionals*. Morgan Kaufmann by Elsevier, 2014.

Hevner, A. R. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, 19:87–92, 2007.

Hevner, A. R., March, S. T., Park, J., and Ram, S. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

Hirschheim, R. and Klein, H. K. Crisis in the IS Field? A Critical Reflection on the State of the Discipline. *Journal of the Association for Information Systems*, 4:237 – 293, 2003.

Hoetker, G., Swaminathan, A., and Mitchell, W. Modularity and the Impact of Buyer–Supplier Relationships on the Survival of Suppliers. *Management Science*, 53(2):178 – 191, 2007.

Hruby, P., Kiehn, J., and Scheller, C. *Model-Driven Design Using Business Patterns*. Springer - verlag, Berlin and Heidelberg, 2006.

Hull, R. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In Meersman, R. and Tari, Z., editors, *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Lecture Notes in Computer Science, pages 1152–1163. Springer, 2008.

Hunton, J. E. Blending Information and Communication Technology with Accounting Research. *Accounting Horizons*, 16(1):55 – 67, 2002.

Huysmans, P. *On the Feasibility of Normalized Enterprises: Applying Normalized Systems Theory to the High-Level Design of Enterprises*. Ph.d. dissertation of the faculty of applied economic sciences - department of management information systems, University of Antwerp, Antwerp, September 2011.

Huysmans, P. and De Bruyn, P. A Mixed Methods Approach to Combining Behavioral and Design Research Methods in Information Systems Research. In *Proceedings of the 21st European Conference on Information Systems (ECIS), Utrecht, the Netherlands*, 2013.

IASB. IAS 18 Revenue. Standard, 2001.

IASB. IFRS 15 Revenue. Standard, 2014.

ISO. Information technology – Business Operational View – Part 4: Business transaction scenarios – Accounting and economic ontology. ISO/IEC 15944-4:2015. Standard, 2015.

Jacobson, I. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, 1992.

Jans, M., Lybaert, N., and Vanhoof, K. Internal Fraud Risk Reduction: Results of a Data Mining Case Study. *International Journal of Accounting Information Systems*, 11(1):17–41, 2010.

Jans, M., van der Werf, J. M., Lybaert, N., and Vanhoof, K. A Business Process Mining Application for Internal Transaction Fraud Mitigation. *Expert Systems with Applications*, 38(10):13351–13359, 2011.

Jans, M., Alles, M., and Vasarhelyi, M. The Case for Process Mining in Auditing: Sources of Value Added and Areas of Application. *International Journal of Accounting Information Systems*, 14(1):1–20, 2013.

Järvinen, P. Action Research is Similar to Design Science. *Quality & Quantity: International Journal of Methodology*, 41(1):37–54, 2007.

KPMG. The Effects of IFRS on Information Systems. White paper, KPMG LLP, 2008.

Kruchten, P. Editor's Introduction: Software Design in a Postmodern Era. *IEEE Software*, 22(2):16–18, March 2005.

Kuhn Jr., J. R. and Sutton, S. G. Learning from WorldCom: Implications for Fraud Detection through Continuous Assurance. *Journal of Emerging Technologies in Accounting*, 3(1):61–80, 2006.

Lee, G. and Xia, W. Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility. *MIS Quarterly*, 34(1):87–114, 2010.

Lehman, M. M. Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE*, 68(9):1060–1076, september 1980.

Lehman, M. M. and Ramil, J. F. Rules and Tools for Software Evolution Planning and Management. *Annals of Software Engineering*, 11(1):15 – 44, 2001.

Li, S.-H., Yen, D. C., Lu, W.-H., and Chen, T.-Y. The Characteristics of Information System Maintenance: an Empirical Analysis. *Total Quality Management & Business Excellence*, 25(3/4):280 – 295, 2014.

Lientz, B. P. and Swanson, E. B. Problems in Application Software Maintenance. *Communications of the ACM*, 24:763–769, 1981.

Loch, C. H., Terwiesch, C., and Thomke, S. Parallel and Sequential Testing of Design Alternatives. *Management Science*, 47(5):663–678, 2001.

Locke, J. and Lowe, A. XBRL: An (Open) Source of Enlightenment or Disillusion? *European Accounting Review*, 16(3):585 – 623, 2007.

Lu, Y. and Ramamurthy, K. R. Understanding the Link Between Information Technology Capability and Organizational Agility: An Empirical Examination. *MIS Quarterly*, 35 (4):931–954, 2011.

Malhotra, R. and Temponi, C. Critical Decisions for ERP Integration: Small Business Issues. *International Journal of Information Management*, 30(1):28 – 37, 2010.

Mannaert, H. and Verelst, J. *Normalized Systems: Re-creating Information Technology Based On Laws For Software Evolvability.* Koppa, 2009.

Mannaert, H., Verelst, J., and Ven, K. The Transformation of Requirements into Software Primitives: Studying Evolvability based on Systems Theoretic Stability. *Science of Computer Programming*, 76(12):1210 – 1222, 2011.

Mannaert, H., De Bruyn, P., and Verelst, J. Exploring Entropy in Software Systems - Towards a Precise Definition and Design Rules. In *Proceedings of the Seventh International Conference on Systems*, pages 93–99, Saint Gilles, Reunion, 2012a. Proceedings of the Seventh International Conference on Systems.

Mannaert, H., Verelst, J., and Ven, K. Towards Evolvable Software Architectures based on Systems Theoretic Stability. *Software: Practice and Experience*, 42(1):89–116, 2012b.

Mannaert, H., Verelst, J., and De Bruyn, P. *Normalized Systems Theory: Towards a Foundational Theory for Evolvable Design*. Forthcoming, 2016.

Mäntylä, M. V. and Lassenius, C. Subjective Eevaluation of Software Evolvability using Code Smells: An Empirical Study. *Empirical Software Engineering*, 11(3):395–431, 2006.

March, S. T. and Smith, G. F. Design and Natural Science Research on Information Technology. *Decision Support Systems*, 15(4):251 – 266, 1995.

Marshall, C. and Rossman, G. *Designing Qualitative Research*. Sage Publications, 3th edition edition, 2006.

McCarthy, W. E. The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. *Accounting Review*, 57(3):554–578, 1982.

McIlroy, D. Mass Produced Software Components. In Naur, P. and Randell, B., editors, *Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*, 1968.

Meall, L. Can you comply? *Accountancy*, 133(1329):73 – 74, 2004.

Midha, V. and Bhattacherjee, A. Governance Practices and Software Maintenance: A Study of Open Source Projects. *Decision Support Systems*, 54(1):23 – 32, 2012.

Mikkola, J. H. Capturing the Degree of Modularity Embedded in Product Architectures. *Journal of Product Innovation Management*, 23(2):128–146, 2006.

Mirani, R. and Lederer, A. L. An Instrument for Assessing the Organizational Benefits of IS Projects. *Decision Sciences*, 29(4):803 – 838, 1998.

Mookerjee, R. Maintaining Enterprise Software Applications. *Commun. ACM*, 48(11): 75–79, November 2005.

Nadler, D. A. and Tushman, M. L. The Organization of the Future: Strategic Imperatives and Core Competencies for the 21st Century. *Organizational Dynamics*, 28(1):45 – 60, 1999.

Nasir, M. I. and Iqbal, R. Evolvability of Software Systems. Master's thesis, Blekinge Institute of Technology, Ronneby, Sweden, 2008.

Neumann, S. *Strategic Information Systems: Competition Through Information Technologies*. Macmillan College Publishing Company, New York, NY, 1994.

Orton, J. D. and Weick, K. E. Loosely Coupled Systems: A Reconceptualization. *Academy of management review*, 15(2):203–223, 1990.

Overby, E., Bharadwaj, A., and Sambamurthy, V. Enterprise Agility and the Enabling Role of Information Technology. *European Journal of Information Systems*, 15(2):120 – 131, 2006.

Page-Jones, M. *The Practical Guide to Structured Systems Design*. Yourdon Press, New York, USA, 1980.

Panorama Consulting Services. Clash of the Titans 2016: An Independent Comparison of SAP, Oracle, Microsoft Dynamics and Infor. Electronically Published: http://panorama-consulting.com/, 2015.

Parnas, D. L. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45 – 77, 2007.

Pinsker, R. and Li, S. Costs and Benefits of XBRL Adoption: Early Evidence. *Communications of the ACM*, 51(3):47 – 50, 2008.

Porter, A. A. Fundamental Laws and Assumptions of Software Maintenance. *Empirical Softw. Engg.*, 2(2):119–131, February 1997.

Prater, E., Biehl, M., and Smith, M. A. International Supply Chain Agility. *International Journal of Operations & Production Management*, 21(5/6):823, 2001.

Rajlich, V. Software Evolution and Maintenance. In *Proceedings of the on Future of Software Engineering*, pages 133–144. ACM, 2014.

Rashid, A., Wang, W. Y. C., and Dorner, D. Gauging the Differences between Expectation and Systems Support: The Managerial Approach of Adaptive and Perfective Software Maintenance. In *Fourth International Conference on Cooperation and Promotion of Information Resources in Science and Technology, 2009 (COINFO '09)*, pages 45–50, Nov 2009.

Reijers, H., Mendling, J., and Dijkman, R. On the Usefulness of Sub-Processes in Business Process Models. Technical report, BPM Report Center, 2010.

Reijers, H. and Mendling, J. Modularity in Process Models: Review and Effects. In *International Conference on Business Process Management*, pages 20–35. Springer, 2008.

Ritchie, J., Lewis, J., and Elam, G. *Qualitative Research Practice: A Guide for Social Science Students and Researchers*, chapter Designing and Selecting Samples, pages 77–108. Sage Publications, Thousand Oaks,, 2003.

Sambamurthy, V., Bharadwaj, A., and Grover, V. Shaping Agility through Digital Options: Reconceptualizing the Role of Information Technology in Contemporary Firms. *MIS Quarterly*, 27(2):237 – 263, 2003.

Sanchez, R. Modular Architectures in the Marketing Process. *Journal of Marketing*, 63 (4):92 – 111, 1999.

Sanchez, R. and Mahoney, J. T. Modularity, Flexibility, and Knowledge Management in Product and Organization Design. *Strategic Management Journal*, 17:63 – 76, 1996.

SAP AG. SAP Documentation. Electronically Published: http://help.sap.com, 2013.

Schilling, M. A. Toward a General Modular Systems Theory and its Application to Interfirm Product Modularity. *Academy of management review*, 25(2):312–334, 2000.

Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., and Lindgren, R. Action Design Research. *MIS Quarterly*, 35(1):37–56, 2011.

Serna, E. M. and Serna, A. A. Ontology for Knowledge Management in Software Maintenance. *International Journal of Information Management*, 34(5):704 – 710, 2014.

Simon, H. A. The Architecture of Complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482, 1962.

Simon, H. A. *The Sciences of the Artificial*. The MIT Press, second edition, 1996.

Sinnett, W. M. and Willis, M. The Time Is Right for Standard Business Reporting. *Financial Executive*, 25(9):23 – 27, 2009.

Smart Business Editiorial Office, editor. Belgian IT Report 2012: Software. Periodical, September 2012.

Smith, H. *Business Process Management: The Third Wave*. Meghan Kiffer, Tampa, Fla, USA, 2003.

Sonnenberg, C. and vom Brocke, J. Evaluations in the Science of the Artificial - Reconsidering the Build-Evaluate Pattern in Design Science Research. In Peffers, K., Rothenberger, M., and Kuechler, B., editors, *Design Science Research in Information Systems. Advances in Theory and Practice*, volume 7286 of *Lecture Notes in Computer Science*, pages 381–397. Springer Berlin Heidelberg, 2012.

Stake, R. *Handbook of Qualitative Research*, chapter Case Studies, pages 236–247. Sage Publications, Thousand Oaks,, 1994.

Stake, R. *Strategies of Qualitative Inquiry*, chapter Case Studies, pages 86–109. Sage Publications, Thousand Oaks,, 1998.

Steele, D. The IT Implications of IFRS. Technical report, Grant Thornton, 2009.

Stoffels, B., editor. ICT jaarboek 2016-2017. Smart Biz, Periodical, July-August 2016.

Stufflebeam, D. L. The CIPP Model for Evaluation. In Kellaghan, T. and Stufflebeam, D. L., editors, *International Handbook of Educational Evaluation*, page 3162, Dordrecht, The Netherlands, 2003. Kluwer Academic Publishers.

Sun, X., Li, B., Leung, H., Li, B., and Li, Y. MSR4SM: Using Topic Models to Effectively Mining Software Repositories for Software Maintenance Tasks. *Information and Software Technology*, 66:1 – 12, 2015.

Tallon, P. P. and Pinsonneault, A. Competing Perspectives on the Link Between Strategic Information Technology Alignment and Organizational Agility: Insights from a Mediation Model. *MIS Quarterly*, 35(2):463–486, 2011.

The Standish Group International Incorporated. Chaos Manifesto 2013 - Think Big, Act Small. Technical report, The Standish Group International Incorporated, 2013.

Ulrich, K. The Role of Product Architecture in the Manufacturing Firm. *Research Policy*, 24(3):419 – 440, 1995.

Van Nuffel, D. *Towards Designing Modular and Evolvable Business Processes*. Ph.d. dissertation of the faculty of applied economic sciences - department of management information systems, University of Antwerp, 2011.

Van Nuffel, D., Mannaert, H., De Backer, C., and Verelst, J. Towards a Deterministic Business Process Modeling Method Based on Normalized Systems Theory. *International Journal on Advances in Software*, 3(1-2):54–69, 2010.

Vanhoof, E. Normalized Accounting Information Systems. Masterthesis, University of Antwerp, 2010.

Venable, J., Pries-Heje, J., and Baskerville, R. A Framework for Evaluation in Design Science Research. *Eur J Inf Syst*, 25(1):77–89, 2014.

Visterin, W., editor. ICT jaarboek 2013-2014. Smart Business, Periodical, July-August 2013.

Visterin, W., editor. ICT jaarboek 2014-2015. Smart Business, Periodical, July-August 2014.

Visterin, W., editor. ICT jaarboek 2015-2016. Smart Business, Periodical, July-August 2015.

Weill, P., Subramani, M., and Broadbent, M. Building IT Infrastructure for Strategic Agility. *MIT Sloan Management Review*, 44(1):57, 2002.

Worren, N., Moore, K., and Cardona, P. Modularity, Strategic Flexibility, and Firm Performance: a Study of the Home Appliance Industry. *Strategic management journal*, 23(12):1123–1140, 2002.

Yigit, A. S. and Allahverdi, A. Optimal Selection of Module Instances for Modular Products in Reconfigurable Manufacturing Systems. *International Journal of Production Research*, 41(17):4063–4074, 2003.

Yin, R. K. *Case Study Research : Design and Methods*. Applied social research methods series. - Newbury Park, Calif.; vol. 5. Sage Publications, Newbury Park, California, 3rd edition/4th edition edition, 2003.