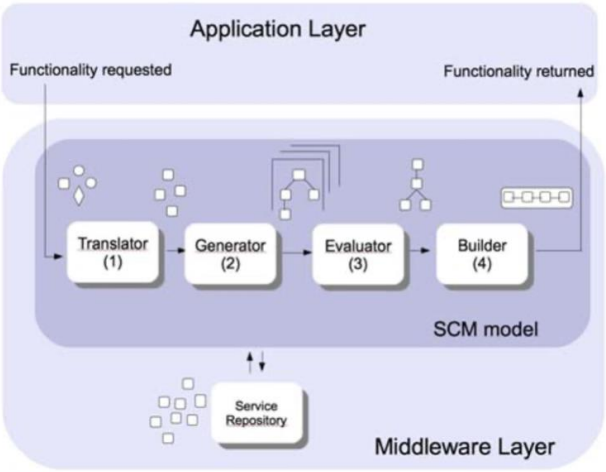
IV. MODEL SREDNJEG SLOJA ZA ORKESTRACIJU SERVISA

Nasuprot apstraktnom okviru i UML4SOA koji se fokusiraju na orkestraciju bez direktnog deklarisanja elementa koji će je voditi, SCM (*Service Composition Middleware Model*) je pristup orkestraciji zasnovan na postojanju srednjeg sloja koji ima ulogu da generiše složene servise i nadgleda njihovo izvršavanje. Zamišljen je kao apstraktni sloj, koji posmatra servise kao "crne kutije" i isti generalizovan pristup orkestriranja primenjuje bez obzira na njihovu konkretnu implementaciju [7]. On je u direktnoj interakciji sa aplikativnim slojem, i predstavlja posrednika u komunikaciji između korisničke i serverske strane. Samim tim, servisi ne odgovaraju na zahteve korisnika direktno, već prepuštaju srednjem sloju da ih presretne i prvi obradi. Na zahteve za funkcionalnostima koje dobije, broker odgovora pozivom servisa, bilo atomičnih koji se nalaze u njegovom registru servisa bilo kompozitnih.

SCM je podeljen na 4 komponente (*Translator, Generator, Evaluator, Builder*) koje učestvuju u izgradnji složenih servisa, pa se i sam proces kompozicije može razložiti na 4 koraka:

1. Aplikacije specificiraju funkcionalnosti koje su im potrebne i upućuju zahtev srednjem sloju. Komponenta prevodilac (*Translator*) preuzima zahtev i prevodi ga u jezik razumljiv brokeru.
2. Preveden zahtev prepušta *Generatoru*. *Generator* treba na osnovu raspoloživih servisa da napravi jedan ili više planova izvršavanja (mogućih kompozicija) i na taj način ponudi tražene funkcionalnosti. Plan slaganja, kao rezultat rada *Generatora*, sastoji se iz lanca interfejsa povezanih na osnovu sintaksičkih pravila ili nekim semantičkim metodama, što može biti prikazano grafički ili opisano jezikom posebno definisanim za tu namenu.
3. *Evaluator* bira najbolji od nastalih planova u zavisnosti od okruženja. Iako algoritam po kome bi se selektovao najoptimalniji plan nije izložen, naznačeno je da implementacija treba da uzme u obzir kontekst aplikacije, modela servisa, aktivnost na mreži…
4. *Builder* je zadužen za izvršenje selektovanog plana i komunikaciju sa konkretnim servisima da bi bio dostavljen rezultat složene funkcionalnosti.

Na slici 4.1. je prikazan generalni model SCM srednjeg sloja. Prednost ovog rešenja je visoka fleksibilnost, jer je u osnovi ideja za projektovanje srednjeg sloja. Radi samo sa dinamičkom orkestracijom, što je korisno za današnje sisteme čija je priroda promenljiva. Ipak, može se reći da je rešenje suviše apstraktno, jer model ne opisuje način interakcije sa okruženjem, niti način čuvanja konteksta i mehanizme reagovanja na njegove promene - to je prepušteno projektantu da reši u fazi dizajna sistema.



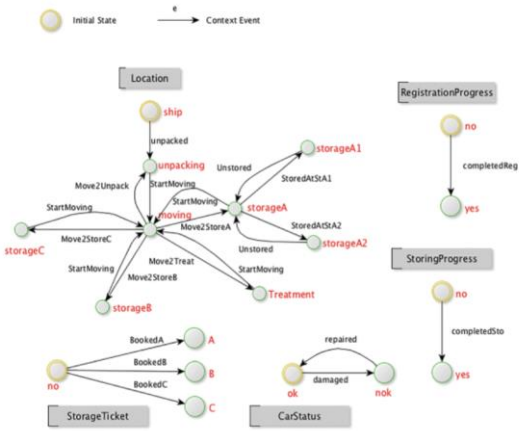
Sl. 4.1. SCM arhitektura.

V. KONTEKST-ZAVISAN OKVIR ZA DINAMIČKO SLAGANJE PROCESA

Jedno od novijih istraživanja fokusira se na kompoziciju fragmenata procesa umesto konkretnih servisa. Fragment predstavlja ponovno upotrebljivi deo procesa ili generalno znanje o njemu koje se može iskoristiti u stvaranju složenih procesa [8]. Ukoliko kompoziciju servisa posmatramo kao složeni proces koji treba izvršiti, ideja iza ovoga je da se konkretni procesi i implementacije mogu menjati u zavisnosti od konteksta, ali da se njihov kostur može iznova koristiti.

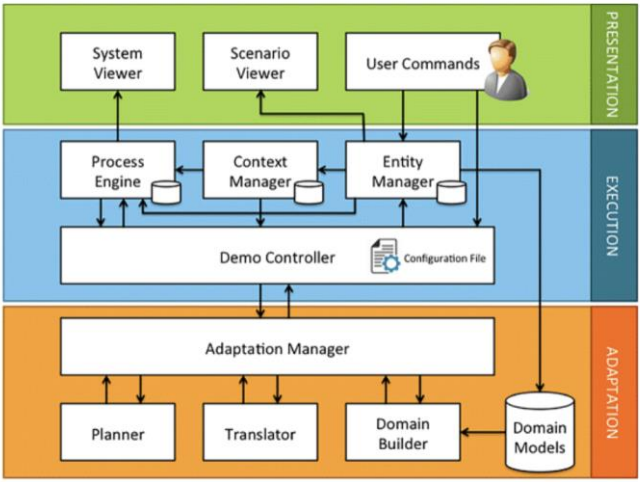
Bućiarone i saradnici su dali novi pristup orkestraciji koji koristi APFL (*Adaptable Pervasive Flows Language*), proširenje BPEL-a, za modelovanje procesa i zasniva se na modelovanju apstraktnih aktivnosti. Apstraktne aktivnosti su zadaci koje može da modeluje čovek, a apstrakcija postoji jer konkretna implementacija direktno zavisi od konteksta. Drugim rečima, modelovanje pokriva stvaranje procesa samo delimično: uzima u obzir korake procesa i to kako bi generalno trebalo da izgleda njegov tok, međutim kako će stvarno biti izvršen zadatak odlučuje orkestrator u toku izvršavanja. On ima sposobnost da, u zavisnosti od konteksta, dinamički menja apstraktno modelovanu aktivnost odgovarajućom kompozicijom fragmenata.

Sa stanovišta modelovanja, fragmenti se predstavljaju kao sistem obeleženih prelaza, što fragment-orijentisane procese svodi na automat. Na slici 5.1. je ilustrovan primer parkiranja auta, prikazan kroz stanja koja su u stvari fragmenti ovog složenog procesa. Prelazi između različitih stanja su dati kao događaji okruženja, gde se vidi da rad svakog fragmenta ostavlja trag u konteksu, tj. da su događaji zapravo posledice njihovog izvršenja. Svaki fragment se može dodatno opisati preduslovom i efektom: preduslov govori iz kojih stanja konteksta je njegovo izvršavanje dozvoljeno, dok efekat opisuje događaje koje će fragment izazvati nakon što se izvrši, a ti dotatni opisi su u formi anotacije na osnovu atributa iz okruženja.



Sl. 5.1. Primer parkiranja auta u fragment-orijentisanom sistemu.

Sam radni okvir (*framework*) se zasniva na proširenju *ASTRO-CAptEvo*[9] okvira, čija je višeslojna arhitektura prikazana na slici 5.2. Dodatne funkcije su ubačene na nivou adaptacije (*Adaptation*) - taj sloj prihvata apstraktnu aktivnost i cilj izvršavanja zajedno sa informacijom o kontekstu, i treba da od njih kreira orkestraciju sa konkretnim fragmentima i planovima izvršavanja. Taj element se potom prosleđuje višim slojevima na obradu, da bismo na kraju dobili tačnu implementaciju procesa koji se na početku posmatrao.



Sl. 5.2. *ASTRO-CAptEvo* radni okvir.

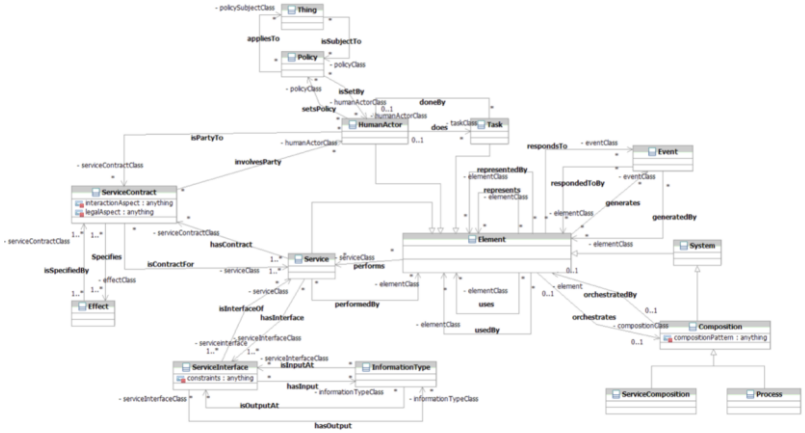
Možemo zapaziti da, kao i u SCM pristupu, postoje komponente za prevođenje i planiranje. Dok je glavna uloga *Planner*-a vrlo slična onoj koju predlaže SCM rešenje - da kreira plan potencijalnih orkestracija na osnovu okruženja - prevodilac se ne bavi obradom zahteva koji dolaze iz spoljašnjeg sveta, već je njegova primarna aktivnost prevođenje problema iz domena kompozicije u domen planiranja. Dodatno, nakon što planer završi sa svojom

obradom, rezultat se vraća prevodiocu da ga iz plana pretvori u proces koji može da se izvrši.

Ovaj pristup kompoziciji je vrlo kompleksan ali uspeva da obuhvati dinamičku orkestraciju zahvaljujući razlaganju složenih servisa i procesa na manje celine i njihovom ponovnom uklapanju u toku izvršavanja. Ne samo da uključuje stanja okruženja, već pruža mogućnost izmene procesa u zavisnosti od konteksta u toku rada sistema, što daje podlogu za kreiranje servisa zavisnih od stanja (*stateful*). S obzirom na to da su mikroservisne arhitekture danas najčešće zasnovane na REST (*Representational state transfer*) protokolu koji je po definiciji bez stanja, ovakav pristup je potpuno nov pogled ne samo na orkestriranje, već i na pisanje servisa. *ASTRO-CAptEvo* daje osnovu za modelovanje samo-adaptirajućeg srednjeg sloja, a u kombinaciji sa servisima koji čuvaju informaciju o stanju bi mogao da u potpunosti promeni koncept SOA.

VI. THE OPEN GROUP META-MODEL ZA RAZVOJ SERVISNIH ARHITEKTURA

U osnovi ovog meta-modela (slika 6.1.) se nalaze klase *Element* i *System*, na čijim temeljima su zasnovani svi glavni gradivni elementi SOA. Činjenica je da se u literaturi često koristi termin komponenta umesto element, pa se oni i u ovom modelu mogu posmatrati kao sinonimi. *Element* je zatvorena jedinica sistema, koja je nedeljiva celina na posmatranom nivou apstrakcije. Detaljno su razmotrene 4 specijalizacije elementa u kontekstu SOA: servis, sistem, akter i zadatak (opisani klasama *Service*, *System*, *HumanActor* i *Task*, respektivno). Za elemente su karakteristične asocijacije korišćenja, odnosno interakcije. Pod vezama *used* i *usedBy* se uglavnom misli na direktnu interakciju između dva elementa (na primer, neko koristi servis), ali se ovom vezom mogu opisati i indirektne interakcije poput pripadanja sistemu (sistem koristi element) i jaka spregnutost između elemennata u kompoziciji.



Sl. 6.1. The Open Group meta-model za razvoj servisnih arhitektura

Kompozicija je izdvojena kao potklasa sistema i napravljena je razlika između složenih servisa i procesa: prema definiciji ovog modela složeni servisi su nastali kombinovanjem servisa dok su u u procese uključeni i akteri i zadaci. Ono što je karakteristično za OG tehnički standard je da je kao parametar kompozicije postavljen podatak o šablonu po kome se slaže. Šablon se odnosi na pristupe slaganju - orkestraciju, koreografiju i kolaboraciju, opisane u uvodnom poglavlju. Ne postoji ograničenje nad vrednošću ovog polja, dokle god ono ima uputstvo kako se slažu elementi koji učestvuju u kompoziciji.

Ako posmatramo samo slučaj orkestracije, ovaj šablon je podržan vezom između kompozicije i elementa. Naime, već smo definisali da ukoliko je reč o orkestraciji, mora da postoji orkestrator, odnosno element koji će nadgledati orkestraciju. U jednom smeru, kompozicija ima najviše jedan element koji je orkestrira, i kardinalitet je 1 samo u slučaju kada je reč o orkestraciji. U suprotnom smeru, element može da orkestrira najviše jednu kompoziciju, u kom slučaju ona mora imati orkestraciju zadatu kao šablon. Treba istaći da,

iako je *Service* potklasa *Elementa*, u praktičnoj primeni njena instanca ne može da ima ulogu orkestratora.

ServiceComposition je rezultat uvezivanja servisa, i prirodno potklasa kompozicije. Kao što je uobičajeno, za njenu implementaciju se koristi *Composite* šablon. Treba istaći da instanca *ServiceComposition* nije sama po sebi složeni servis (*Service* i *System* su disjunktne klase), već je element koji izvršava kompozitni servis, tj. ima ulogu orkestratora ukoliko je o tom šablonu reč. Primera radi:

- A i B su instance klase *Service*
- X je instanca klase *ServiceComposition*
- X koristi A i B (slaže ih u skladu sa svojim šablonom kompozicije)

Iako vrlo detaljan model servisnih arhitektura, možemo zapaziti da je ponovo reč o isključivo statičkoj orkestraciji. Ono što je prednost u odnosu na druga rešenja je razdvojenost koncepata procesa i orkestracije. Dodatno, akteri su uključeni kao zasebni elementi, što nije slučaj ni u jednom od do sada predstavljenih meta-modela.

VII. DOGAĐAJIMA UPRAVLJANA PLATFORMA ZA ORKESTRACIJU SERVISA

Mikroservisne arhitekture propagiraju visok nivo dinamike i fleksibilnosti, pa BPEL i slični jezici nisu dobar izbor za njihovu orkestraciju jer bi se ona uglavnom izvršavala u okvirima teškog ESB (*Enterprise Service Bus*) [10]. *Medley* je platforma čija je uloga da pojednostavi specifikaciju procesa sačinjenih iz više mikroservisa kao i njihovog pakovanja i izvršavanja na serveru, pri čemu teži da optimizuje upotrebu resursa u odnosu na ESB. Tako definisana, ona preuzima ulogu srednjeg sloja i umesto krajnjeg korisnika komunicira sa udaljenim servisima.

Rad sa *Medley* okvirom je zamišljen tako da korisnik opiše koje servise treba aktivirati u zavisnosti od kog događaja, i ta specifikacija se prepušta *Medley* kompajleru da generiše kod koji će omogućiti komunikaciju između pojedinačnih elemenata. Kada je orkestracija generisana, svaki servis dobija portove za osluškivanje i ispaljivanje događaja (*publish/subscribe* mehanizam). Pored toga, kompajler je generisao set pravila prepisivanja: svaki

servis kada dobije događaj, ima informaciju o tome kako će da ga preimenuje i objavi kao nov događaj.

Specifikacija orkestracije je zasnovana je na posebno kreiranom domen-specifičnom jeziku pomoću koga korisnik može da opiše proces, odnosno redosled poziva servisa, pošto se servisi mapiraju na procese. Proces je još izražen i kao serija događaja. Ono što ovaj koncept čini dinamičkim je to što jezik dopušta paralelne i asinhrone pozive, kao i definisanje bazena servisa iz kojeg će se birati onaj koji se poziva. To je pogodno ako neki servis nije dostupan u momentu poziva, ako je veza ostvarena sa njim loša ili ako treba da odgovori na preveliki broj poziva.

VIII. ZAKLJUČAK

Na osnovu rešenja izloženih u ovom radu, može se zaključiti da se, iako je statička orkestracija i dalje prisutna u modelima, savremena istraživanja više baziraju na dinamičkim pristupima. Distribuirani sistemi i usluge se sve češće nalaze u promenljivim okruženjima, i potrebno je odgovoriti na događaje u realnom vremenu. Potreba za uključivanjem konteksta u rad složenih sistema se javlja prevashodno zbog sve veće spregnutosti funkcionisanja programa i uređaja sa dešavanjima u spoljašnjoj sredini.

Ovo se ne odnosi samo na SOA, već važi i za mikroservisne arhitekture i za IoT. Zapravo, sa napretkom ovih paradigmi, pojavio se i nov koncept koji počinje da uzima maha - IoS (*Internet of Services*) koji nalaže da sve što je potrebno za korišćenje softvera treba da bude dostupno u formi servisa na internetu, uključujući softver, alate za njegov razvoj, platforma na kojoj će se nalaziti. Takav pristup zahteva nove načine modelovanja, koji nisu isključivi, i podržavaju postojanje samo-adaptirajućih komponenti. One su takozvani "pametni" elementi sistema, koji znaju da raspoznaju stanja u kojima se sistem nalazi i promene svoje izvršavanje tako da bude u skladu sa tim. Zato, kada je reč o modelom-upravljanom razvoju servisnih arhitektura, poželjno je da model ne bude striktno vezan za jedan od ovih koncepata, već da podrži integraciju heterogenih komponenti bez osvrta na to da li se one deklarišu kao servis, mikroservis ili uređaj, dokle god se njihove funkcije mogu pozivati preko interfejsa i posmatrati po principu "crne kutije".

UML4SOA i FOCAS su meta-modeli i daju set uputstva za definisanje orkestracije. *Medley* platforma sa druge strane uvodi dinamično slaganje, međutim ima relativno ograničen opis događaja (ime, podaci koji se prenose, hijerarhija poziva iz okruženja). SCM i ASTRO-CAptEvo kao radni okviri daju najviše fleksibilnosti, ali su ujedno i najzahtevniji sa stanovišta implementacije.

Ono što je karakteristično za sva pomenuta rešenja je svakako način predstavljanja složenih servisa. U većini slučajeva, orkestracija je opisana kao proces koji se sastoji iz mnoštva nezavisnih aktivnosti i postoji komponenta kojoj je dodeljeno da upravlja njenim izvršavanjem. Dinamičke reprezentacije servisno-orijentisanih okruženja su mahom zasnovne na događajima (*event-driven*), i servisi su istaknuti kao komponente koje ujedno i generišu događaje i odgovaraju na njih. Ovo je u formi takozvanog *publish-subscribe* mehanizma, koji u UML klasnom modelovanju može biti iskazan *Observer* šablonom.

Još jedan od ključnih elemenata koji treba da se uključi u rešavanje problema dinamičke orkestracije je okruženje, odnosno kontekst. Na osnovu sagledanih rešenja, dolazi se do zaključka da to treba uraditi formiranjem modela konteksta koji će čuvati opise ili događaja do kojih dolazi ili samih servisa u različitim trenucima, nešto slično preseku stanja. U slučaju da rešenje koje se gradi treba da bude u formi radnog okvira, tada je potrebno zaokružiti ga domen-specifičnim jezikom i algoritmima koji će interpretirati opise kompozicija napisane na njemu. Ako je reč o meta-modelu, dovoljno je definisati konstrukte i pravila za njihovo povezivanje, tako da rezultat bude jednoznačno uputstvo za slikoviti prikaz orkestracija.

LITERATURA

[1] The Open Group, Service-Oriented Architecture Ontology, Technical Standard, 2010, Available:https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/OPGRP_US/O101028S.pdf.

[2] Dipanjan Chakraborty and Anupam Joshi. Dynamic Service Composition: State-of-the-Art and Research Directions. Technical Report TR-CS-01-19, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Maryland, USA, 2001..

[3] Maja Vuković, Context aware service composition, Technical Report,Cambridge CB3 0FD United Kingdom, 2007.

[4] Philip Mayer, Andreas Schroeder and Nora Koch, ”A Model-Driven Approach to Service Orchestration”, Institute for Informatics, Ludwig-Maximilians-University Munich, Germany, 2008.

[5] Stephanie Chollet and Philippe Lalanda “An Extensible Abstract Service Orchestration Framework”, IEEE International Conference on Web Services 2009.
[6] Gabriel Pedraza and Jacky Estublier, “An Extensible Services Orchestration Framework through Concern Composition”, International Workshop on Non-functional System Properties in Domain Specific Modeling Languages (NFPDSML), 2008.
[7] Noha Ibrahim and Frédéric Le Mouël, “A Survey on Service Composition Middleware in Pervasive Environments”, IJCSI International Journal of Computer Science Issues 2009.
[8] Bucchiarone et al., “A context-aware framework for dynamic composition of process fragments in the internet of services”, Journal of Internet Services and Applications volume, 2017..
[9] Bucchiarone et al, “Dynamic adaptation of fragment-based and context-aware business processes”, IEEE 19th International Conference on Web Services, 2012.
[10] Ben Hadj Yahia, E., Réveillère, L., Bromberg, Y.-D., Chevalier, R., & Cadot, A. “Medley: An Event-Driven Lightweight Platform for Service Composition”, Web Engineering, 2016.

ABSTRACT

Contrary to the monolithic systems which are aware of their internal structure and component organisation, when considering orchestration in the service-oriented architecture (SOA), there is no knowledge of how services are implemented which makes it rather challenging to form the right foundation for their collaboration. One of the approaches for modelling orchestration in distributed systems is to introduce a middleware which would be responsible not only for gathering the services but also for composing them. Of course, the orchestration itself can be considered either static or dynamic, where the dynamic approach stands for the middleware's ability to change the patterns of composition at run-time based on the given context. The aim of this paper is to analyse model-driven solutions for service composition proposed so far and to set grounds for developing a meta-model that would enable modelling context-aware middleware solutions in SOA.

The Comparative Analysis of Model-Driven Service Orchestration Solutions

Ana Marković