



d4SCIENCE

ECDL 2009
Corfu

Adaptable Services (Functional Adaptivity for DL Services in e-Infrastructures)

F. Simeoni, L. Candela, D. Lievens, P. Pagano, M.Simi
University of Strathclyde/ISTI-CNR/Trinity College Dublin

context

- *gCube*, services for scientific infrastructures
- goals, concepts and challenges

functional adaptivity

- general-purpose services and bespoke requirements
- functionality, interpretations and implementations

design space

- interfaces, services, components, plugins

conclusions

- services can have, select and acquire multiple behaviours
- ad-hoc adaptations: services and *tasks*

software for eScience

- the full lifecycle of modern scientific enquiry
- stress on information management, EGEE underneath
- collate, describe, annotate, transform, index, search, present
- services, information, machines as shareable resources

virtual research environments

- service-based applications vs. distributed Digital Libraries
- gCube, a DLMS 'with infrastructure'

past, present, future

- OpenDLib (2000), DILIGENT (2004-2007)
- D4Science (2008-2010), D4Science-II (2010-2012)
- 100+ components, serving EM and FARM communities

core

- VRE, security and processes
- publishing/discovery

organisation

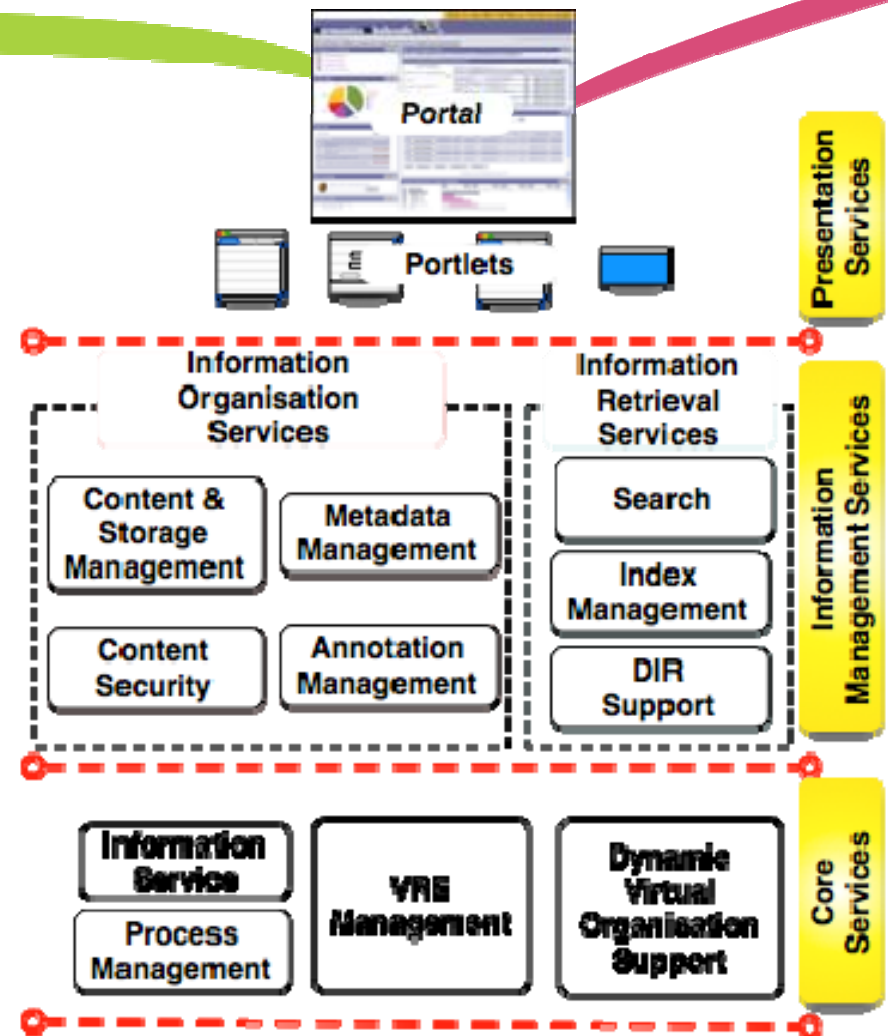
- storage, metadata, annotation
- binary relationship model

retrieval

- indexing, querying, merging
- structured, unstructured

presentation

- portals of pluggable portlets



dynamic service management

- no local administration, no predefined location
- dynamically deployed, where and when convenient
- configuration, staging, scoping, monitoring, orchestration, security
- requirement: state-of-the-art autonomic system

development complexity

- autonomy raises non-trivial requirements on runtime behaviour
- transparencies for developers, not only users!
- *gCore*: dedicated container & application framework
 - systemic aspects of development, best practices
 - sweeping changes for system maintenance and evolution
 - foothold in service design, compile-time solutions

a design dilemma

- serve communities with different requirements
- serve them well, no common denominator solutions
- generality of design vs. specificity of requirements

tools of the trade

- abstract, parameterise, compose
 - exchange data in general-purpose models & formats
 - parameterise interfaces for custom processing of exchanged models
 - compose custom processes into bespoke workflows
- declarative specification of novel behaviour
 - feed specifications to 'engines' (e.g. XSLT, WS-BPEL engine)
- all used in gCube, yet do not cater for all adaptations

DIR services

- *collection selection*: limit evaluation to ‘best’ collections
- *result fusion*: harmonise independently produced rankings
- *collection description*: profile collections for selection and fusion

DIR strategies

- e.g. cooperative vs. uncooperative approaches
- diverge in assumptions, inputs, algorithms and processes
- e.g. statistics extraction vs. query-based sampling

DIR requirements

- wish to support arbitrary strategies
- adaptation beyond customisations, compositions, declarations
- needs *alternative implementations*

development model

- open, infrastructure must remain sustainable
 - no versioning of existing services
- simple, ideally within reach of community expertise

deployment model

- dynamic, no disruption in production

discovery and use model

- uniform, implementations to specialise common interface
 - requires support for **'covariant' subtyping**
- transparent, system to find suitable implementation for client
 - supports a notion of **matchmaking**

implementations as services

- concrete implementations of public interface
- dynamically deployed as any gCube service
- discovered by features of implementation
- matched by dedicated service within infrastructure
 - cf. semantic web, ontologies, description logics

concerns

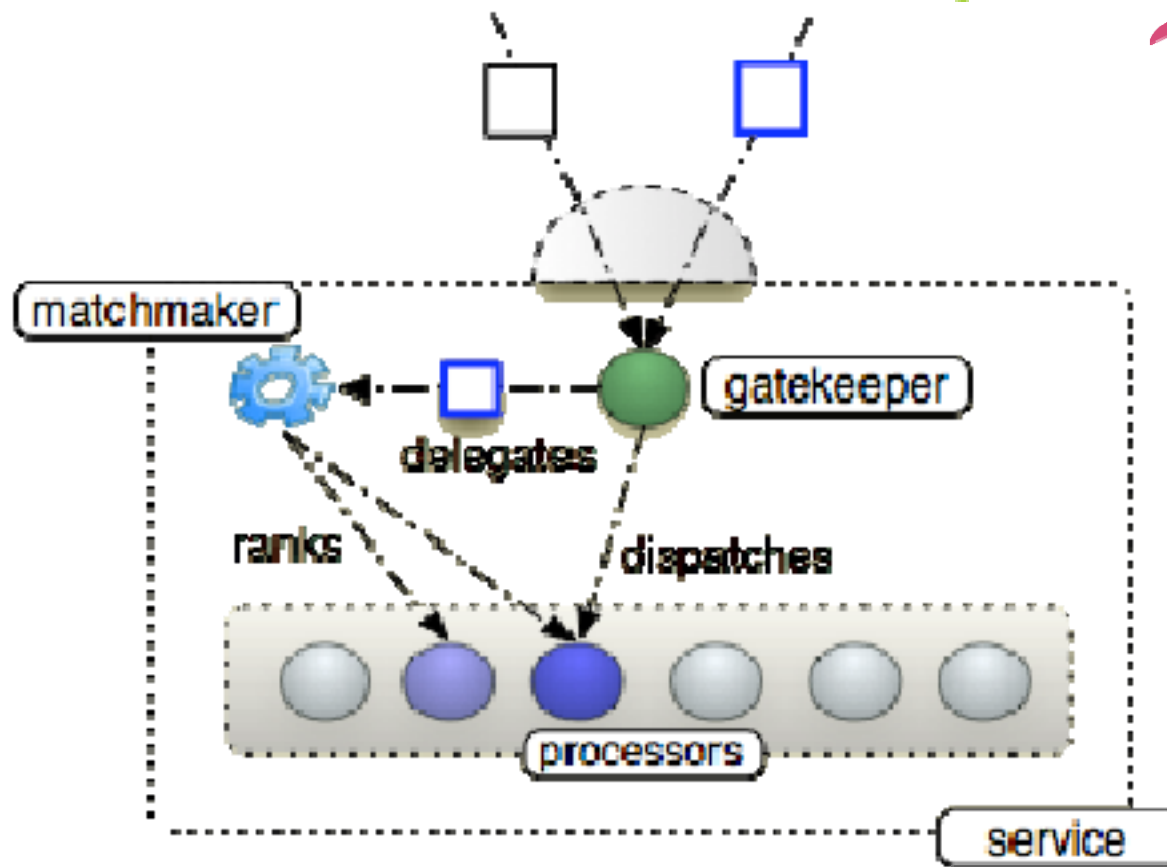
- full-blown service development not always justified
- independent service staging not always desired
- ‘semantic matchmaking’ not fully convincing
 - non-trivial adoption costs
 - state-of-the-art somewhat volatile
 - no evidence of fitness for purpose

implementations as local ‘processors’

- concrete implementations of a private framework
- published as service properties
- ranked by local matchmaker against request
- selected as per service semantics

appeal

- familiar development model, no spurious complexity
- no distributed state management concerns
- request and local knowledge available for very late matching
- selection and dispatch policy under control of service
- smoother and incremental adoption within current system

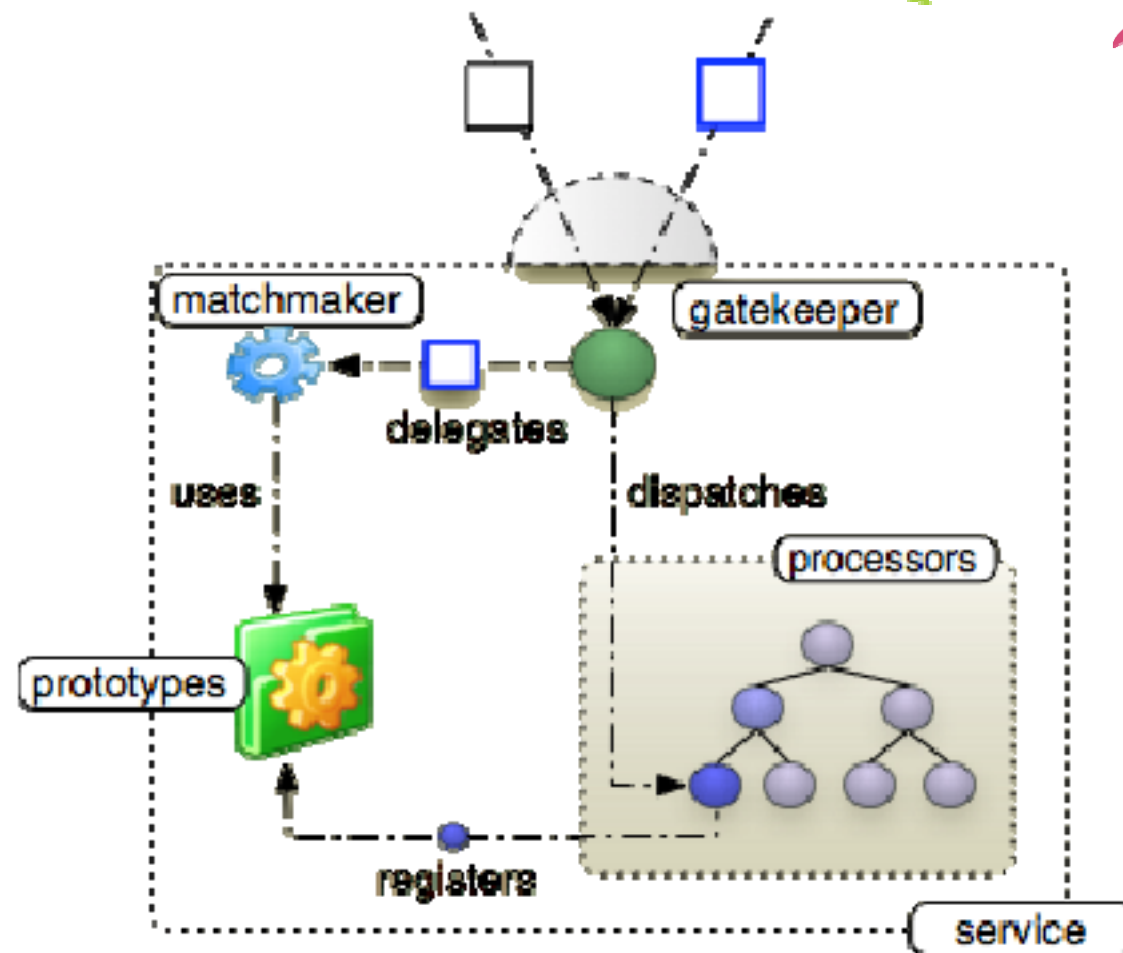


suitability as specificity

- how specifically processors can handle the request
- service interface 'open' to covariant specialisations of input types
- available processors mirrors input specialisations
- processors register prototypical inputs with local matchmaker

matchmaking as subtyping

- deep subtype check between prototype and input object graphs
- 'type distance' along subtype hierarchy determines node score
- a range of metrics to combine scores across sibling nodes
- final score for prototype becomes score for processor
- see related work for details



dynamic deployment

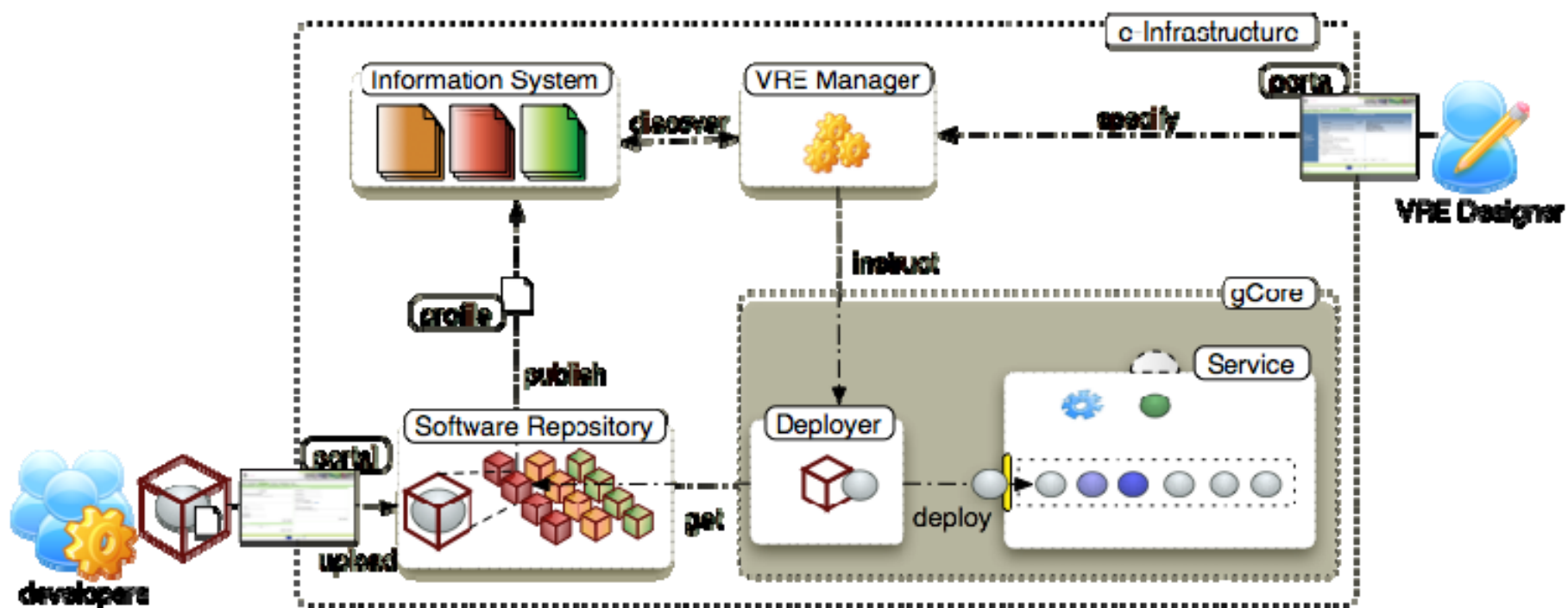
- can services acquire new behaviour after deployment?
- yes, if carried as the payload of service *plugins*

plugins

- standard software libraries
 - code, resources, config, dependencies
- comply with service and infrastructure
 - managed by Core Services at runtime, by gCore at design-time
- a new form of shared resource

plugin lifecycle

- development, packaging, upload, publication
- deployment and undeployment



from services to tasks

- not all requirements fit existing services
 - they call for the execution of ad-hoc *tasks*
- not all tasks fit the service paradigm
 - specific, too lightweight, incompatible with existing design...
 - staging strategies, update procedures, ad-hoc workflow logic....

from tasks to the Executor service

- a container of tasks in a container of services
- lending a public interface for 'monitorable' execution
- publishes, persists, resumes the state of execution
- tasks as service plugins, of course

general design and community needs are in tension

- this threatens scope or adoption

generic functions may need multiple implementations

- these may be hosted and matched *inside* individual services
- services may exhibit polymorphic behaviour

concrete needs may induce ad-hoc functions

- these may be hosted and executed *by* a dedicated service

sharing fine-grained resources can ease the tension

- services may acquire behaviour at runtime
- without compromising sustainability