Configuring e-Government Services Using Ontologies

Dimitris Apostolou

Ljiljana Stojanovic

Tomas Periente Lobo

Jofre Casas Miro

Andreas Papadakis

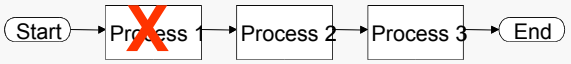
The Problem

Politicians define the law

➔law is revised

Managers decide how to implement the law

➔activities have to be changed



Programmers write code

➔software is to be modified

Citizens deal with eGov services

➔eGov-services are updated

The Goal

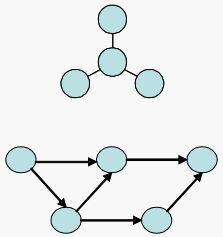
Improve the (back-office) management of e-Gov services

- bridging the gap between decision making and technical realisation of e-Gov services
  - Supporting all back-office phases (design, configure, deploy, run)
- considering the lifecycle of e-Gov services
  - Supporting the management of changes in e-Gov services (preserve consistency, detect inconsistencies, propagate changes, implement changes)
- making knowledge explicit
  - capturing knowledge about e-Gov services
  - tracing design decisions leading to e-Gov service models

The Software

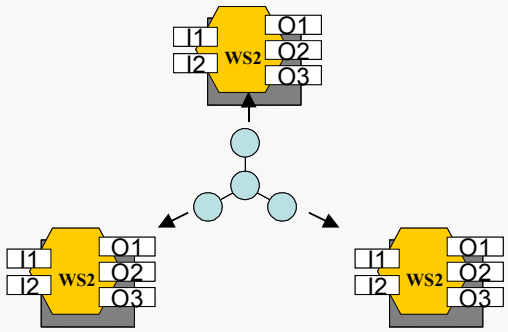
Design Services

Ontology Management System



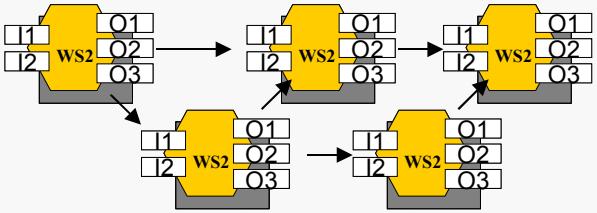
Configure & Deploy Services

Configuration Framework



Run Services

Runtime Framework

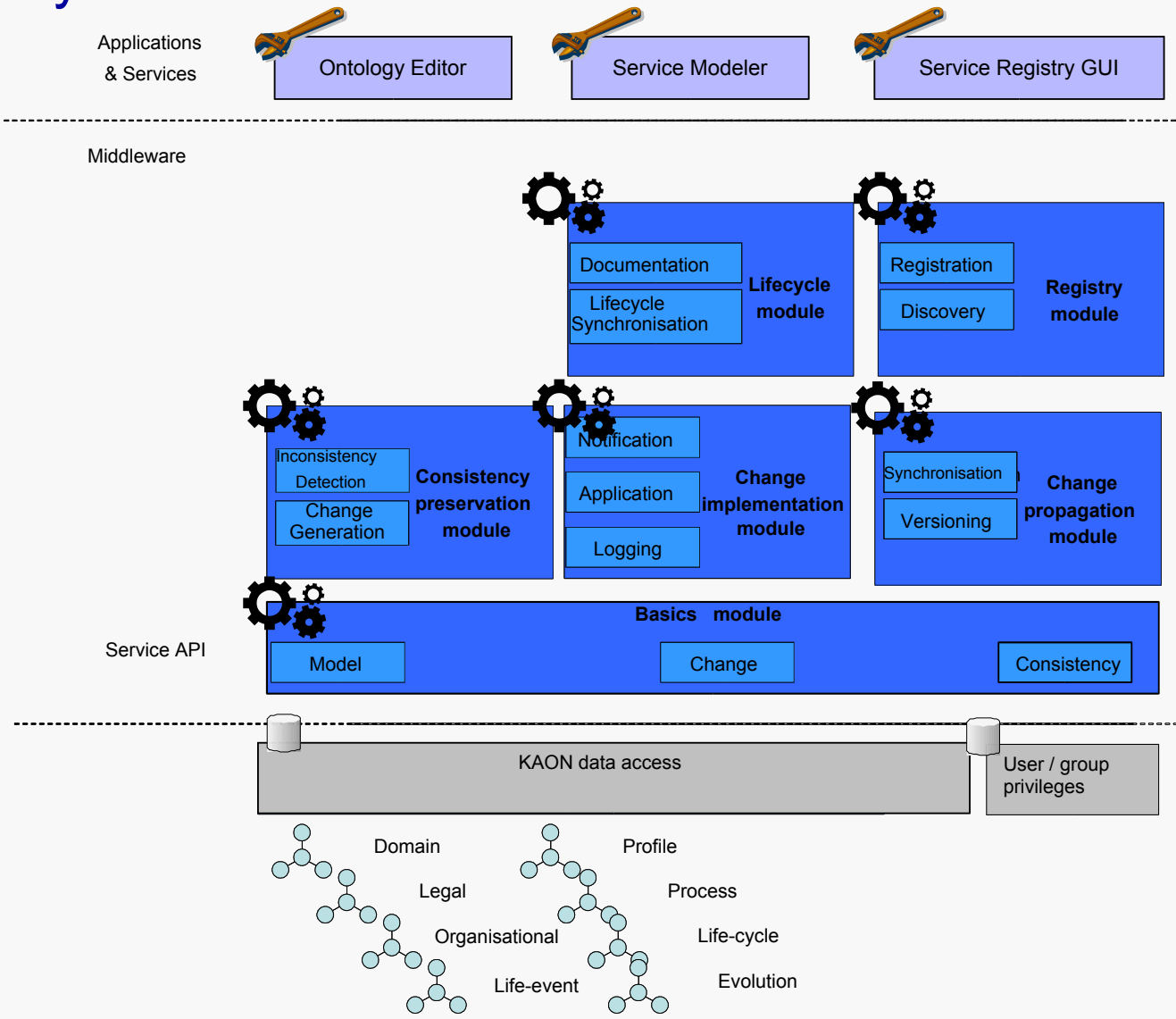


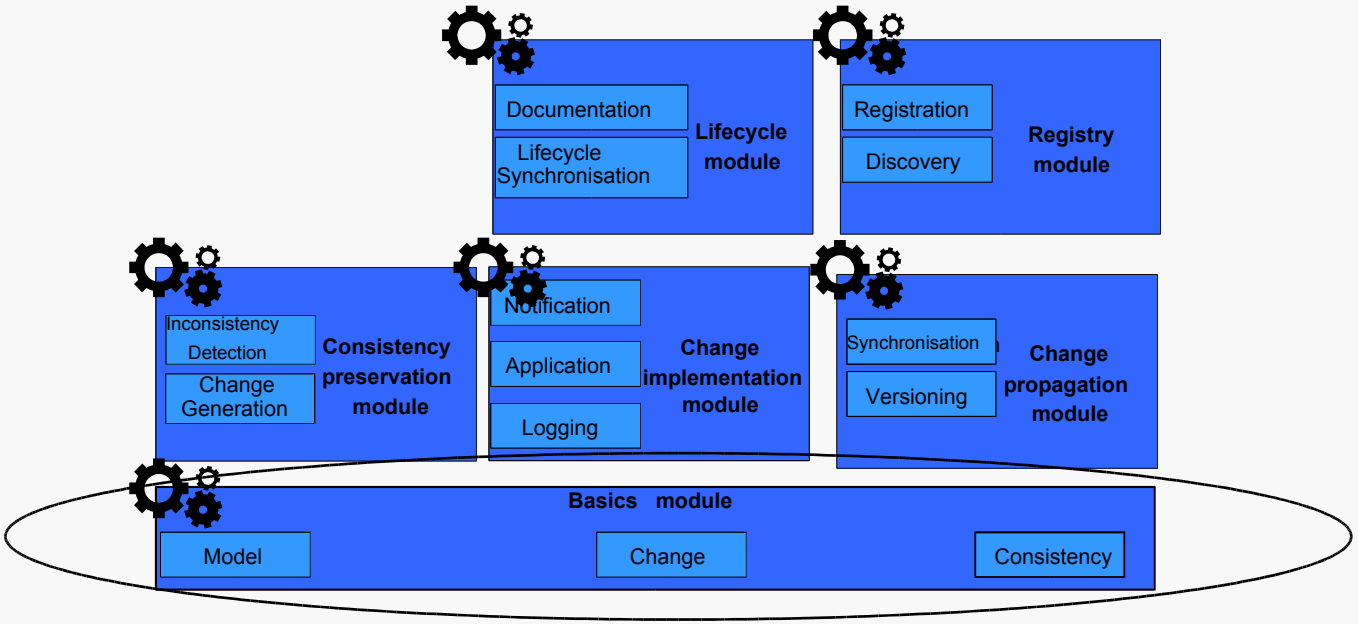
Web services interfaces to legacy systems

What is the OntoGov Ontology Management System?

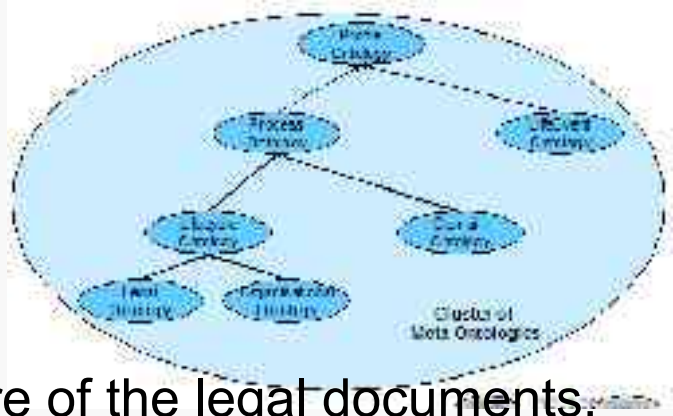
- the OntoGov Ontology Management System creates **ontology-based descriptions of e-Government services**
- It supports the service lifecycle management, which includes
  - service modelling
  - service composition
  - service re-modelling
  - service reuse
  - service discovery

Conceptual architecture of the Ontology Management System



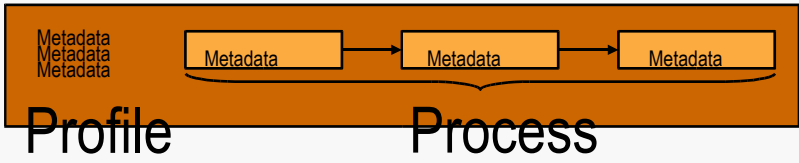

Documentation
Lifecycle Synchronisation
Lifecycle module
Registration
Discovery
Registry module
Inconsistency Detection
Change Generation
Consistency preservation module
Notification
Application
Logging
Change implementation module
Synchronisation
Versioning
Change propagation module
Basics module
Model
Change
Consistency

Models

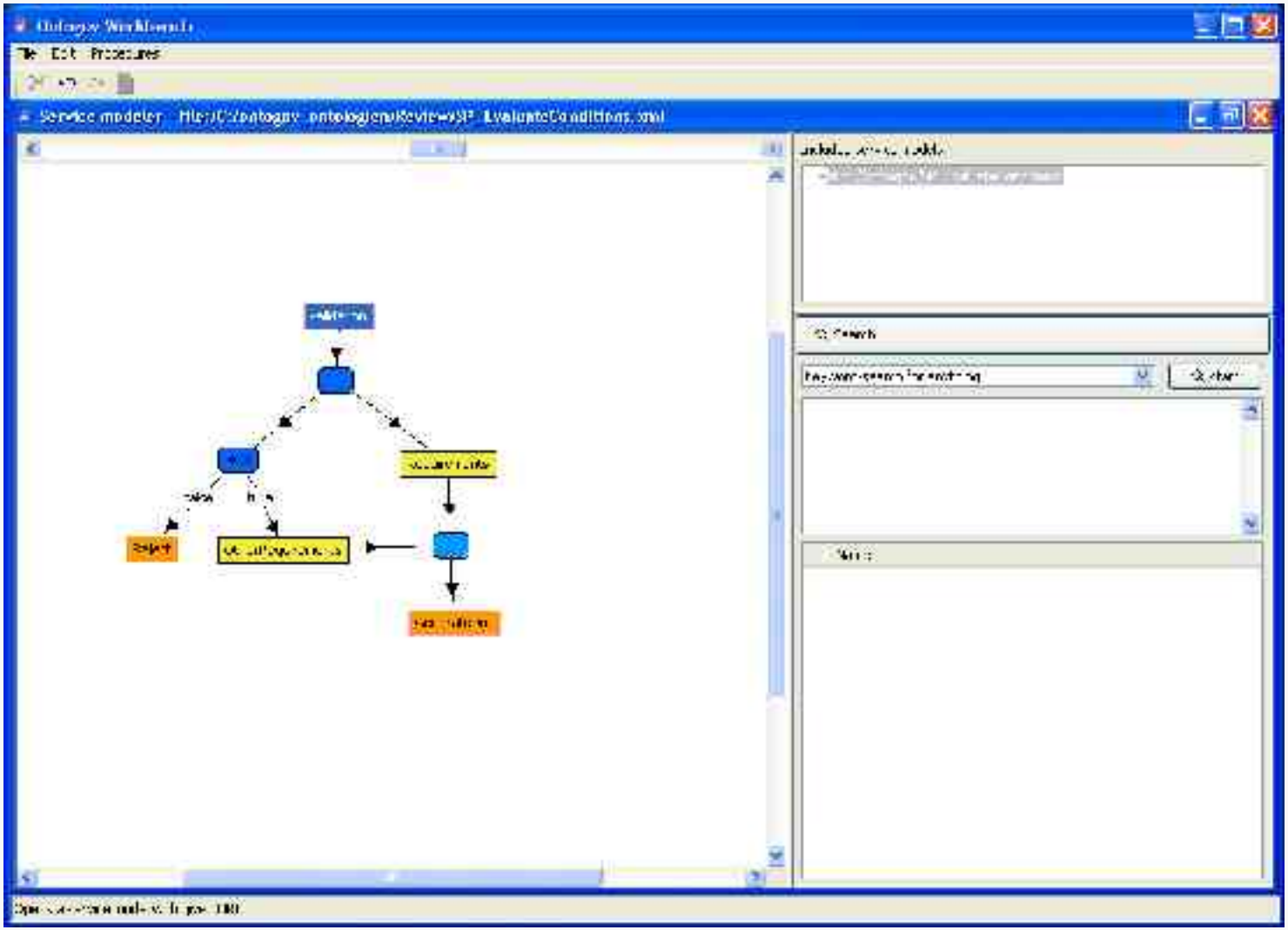


- ***Legal Ontology*** defines the structure of the legal documents, which includes paragraphs, sections, amendments, etc.
- ***Organisational Ontology*** models an organisation by defining its organisational units, roles, persons, resources etc.
- ***Domain Ontology*** contains domain specific knowledge
- ***LifeEvent Ontology*** models the categorisation of the e-Government services
- ***Process Ontology*** describes the elements for modelling the process flow
- ***Profile Ontology*** contains metadata about e-Government services and includes all previously mentioned ontologies
- ***Lifecycle Ontology*** describes the information flow and the decision making process in the public administration

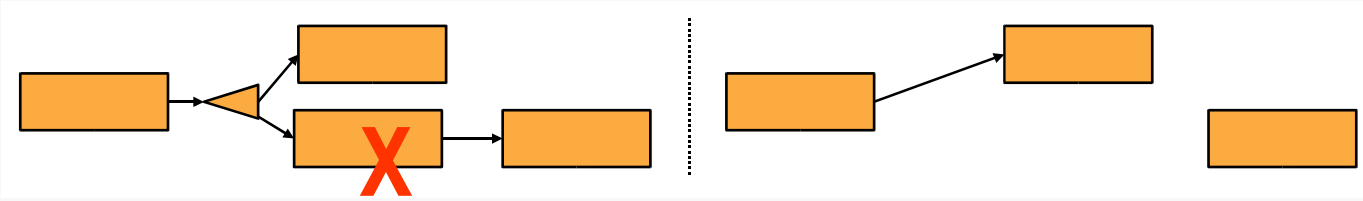
Models: Service Ontology



- It is based on the OWL-S Process Ontology
- We distinguish between the services and the control constructs
- Services can be either atomic or composite services
- We define a standard set of attributes such as name, description, version, status etc.
- There are specific requirements concerning retraceability, realisation, security, costs, etc.
  - each service can be associated to the laws it is based upon
  - each service can be associated to several software components that implement it (i.e. dynamic binding)
  - it is possible to assign security levels to each service
  - information about cost and time restrictions can be also specified



Changes



- To make a service s1 a predecessor of a service s2, a domain expert needs to apply a *list* of ontology changes that connects s1 to s2
- E.g. RemoveAtomicService X
  - Precondition – AtomicService X has been defined
  - Postcondition – AtomicService X doesn‘t exist anymore
  - Actions:
    - Remove all input links of AtomicService X;
    - Remove all output links of AtomicService X;
    - Remove all metadata defined for AtomicService X that includes:
      - the attributes such as name, description, fist and last service;
      - the relations to the legal, organisational and domain ontology;
      - the pre- and post-conditions
- Each change is described in a form:
  - Precondition - a set of assertions that must be true to be able to apply a change
  - Postcondition - a set of assertions that must be true after applying a change and it describes the result of a change
  - Actions are additional changes that have to be generated

Consistency

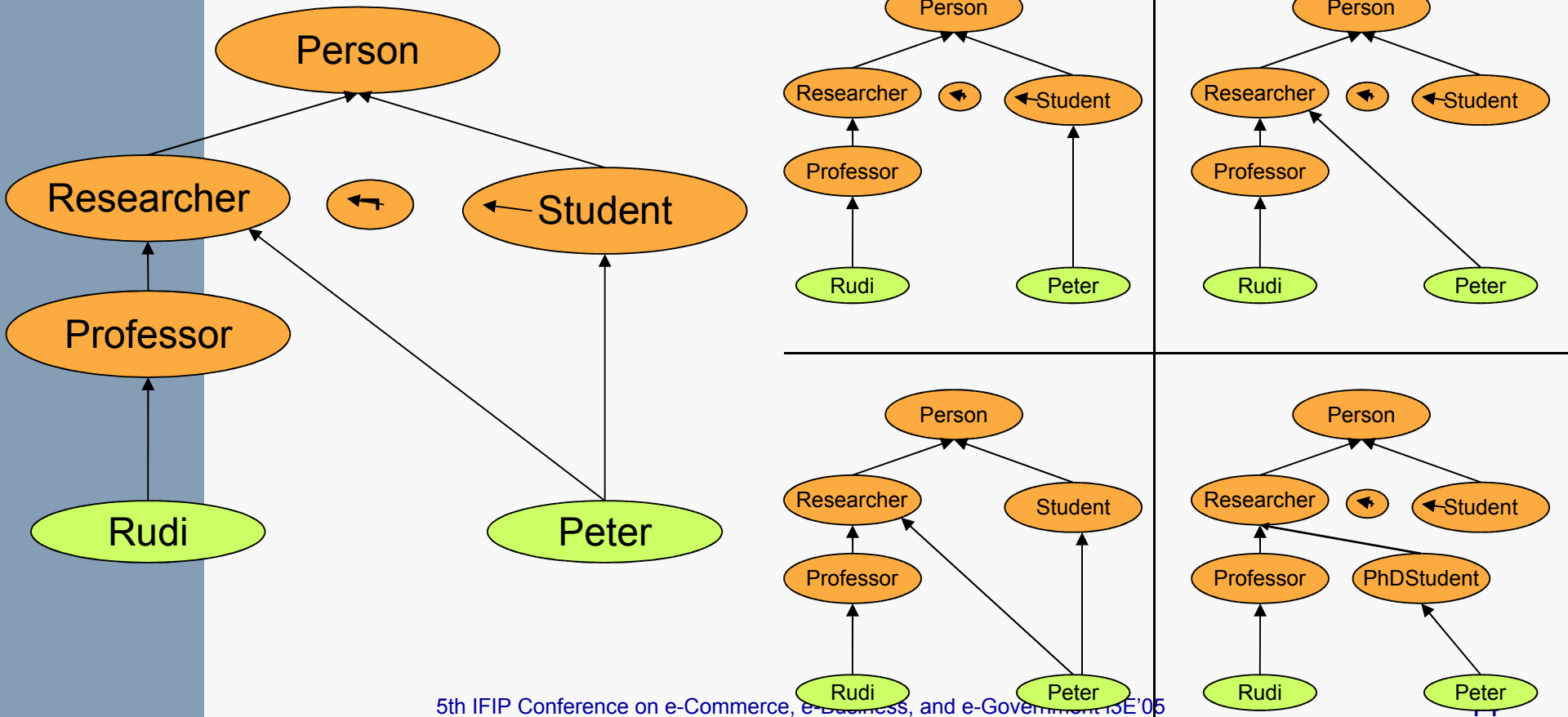
- An e-Government service ontology is consistent:
  - if it is structurally consistent
  - If it is logically consistent
  - if it satisfies a set of user-defined consistency constraints defined for the service model

Consistency

- if it is structurally consistent
  - whether the ontology conforms to certain structural constraints,
  - E.g. OWL Lite disallows the use of negation $\neg C$
  - Simplest solution – removal of axioms that violate constraints
  - Advanced option – express the invalid axiom(s) in a way that is compatible with the defined fragment
    - Syntactic Rewrites (semantically equivalent)
    - Language Weakening / Approximation

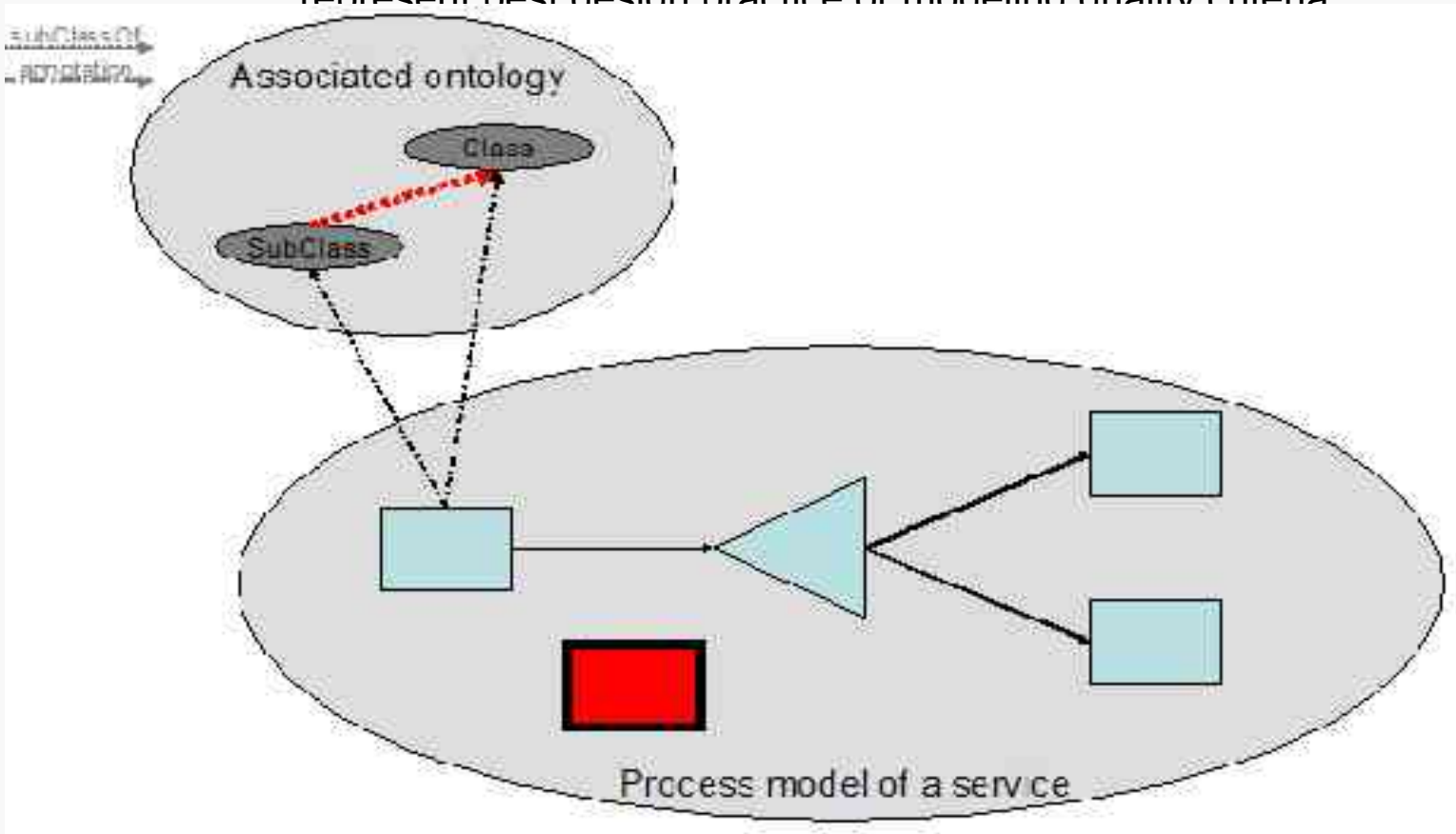
Consistency

- If it is logically consistent
  - whether the ontology is “semantically correct”, i.e. does not contain contradicting information.



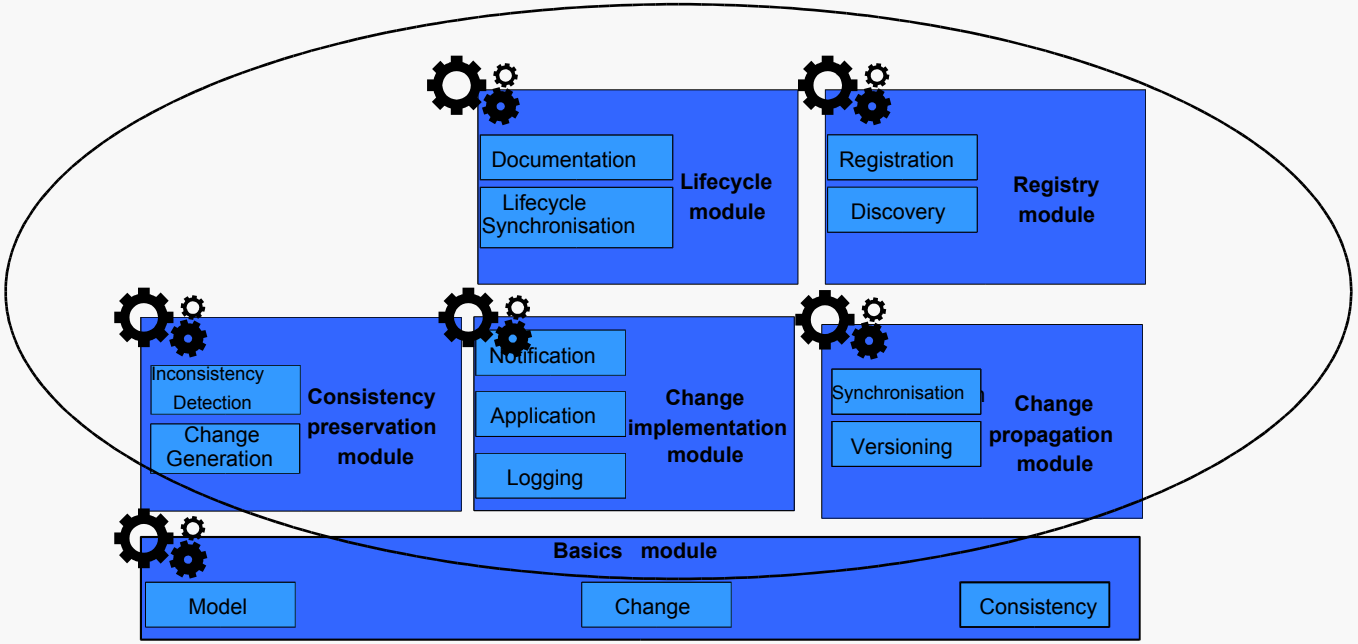
Consistency

- If it satisfies a set of user-defined consistency constraints
- Two types of the user-defined consistency conditions are identified:
  - ***generic conditions*** that are applicable across domains and represent best design practice or modeling quality criteria

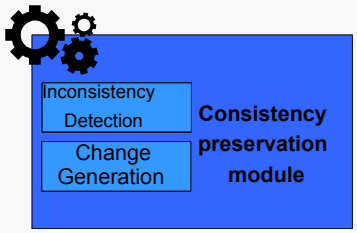


…more consistency checks

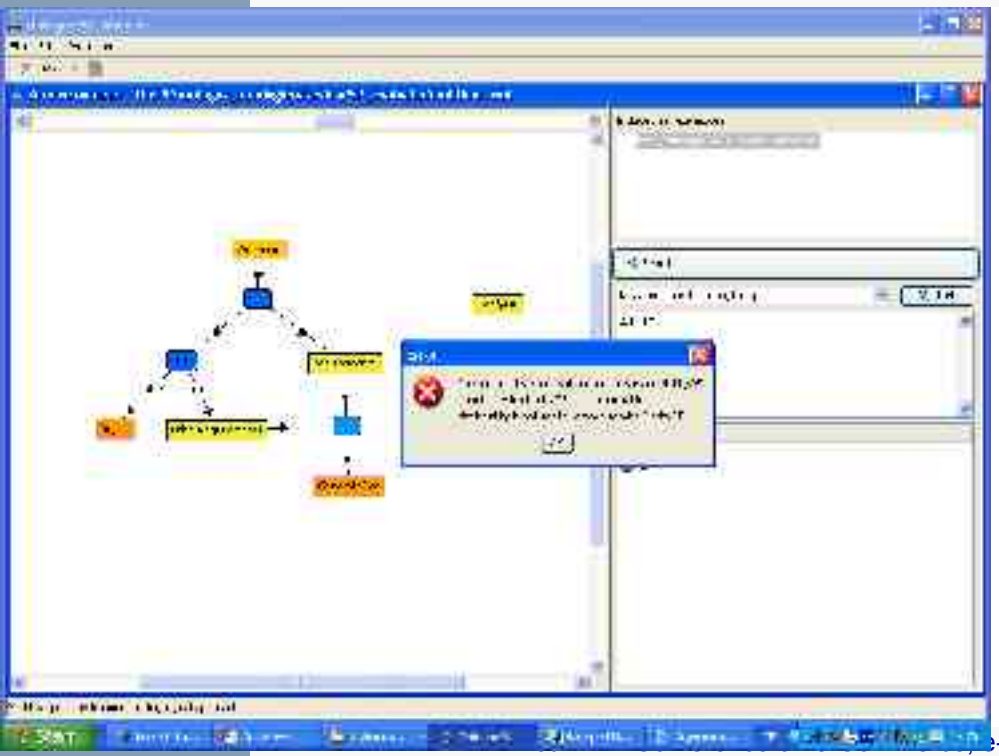
- (C1) Each service has to have a reference to at least one business rule (law).
- (C2) Each service has to have at least one resource that controls its execution.
- (C3) Each service has to have at least one software component attached to it that implements it.
- (C4) Each service has to have at least one input.
- (C5) Each service has to have at least one output.
- (C6) Each service input has to be either output of some other service or is specified by the end-user.
- (C7) If the input of a service is the output of another service, then it has to be subsumed by this output.
- (C8) If the input of a service subsumes the input of the next service, then its preconditions have to subsume the preconditions of the next one.
- (C9) If two services are subsumed by the same service, then their preconditions have to be disjoint.
- (C10) If a service specialises another service, one of its parameters (i.e. inputs, outputs, pre- or post-conditions) has to be different.
- (C11) Any specialization of the activity A1 must always be a predecessor of any specialization of the activity A2, where A1 and A2 are two activities defined in the Meta Ontology and their order is given in advance (i.e. A1 precedes A2).

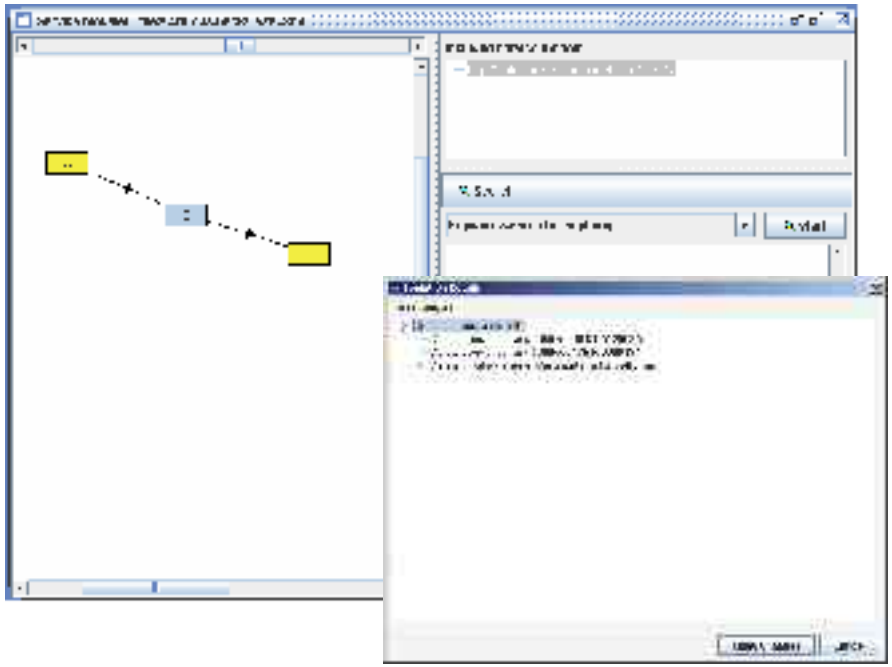

Documentation
Lifecycle Synchronisation
Lifecycle module
Registration
Discovery
Registry module
Inconsistency Detection
Change Generation
Consistency preservation module
Notification
Application
Logging
Change implementation module
Synchronisation
Versioning
Change propagation module
Basics module
Model
Change
Consistency

Consistency preservation

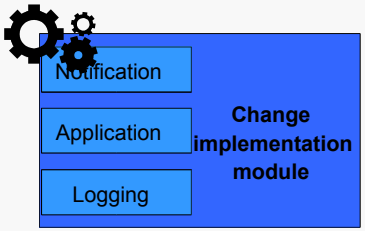


- *Inconsistency Detection*: It is responsible for checking of the consistency of an ontology with the respect to the ontology consistency definition. Its goal is to find ”parts” in the ontology that do not meet consistency conditions
- *Change Generation*: It ensures the consistency of the ontology by generating additional changes that resolve detected inconsistencies



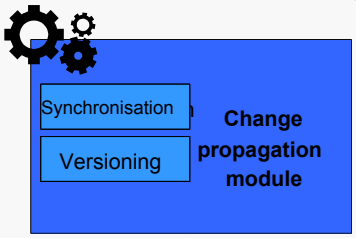


Change Implementation Module

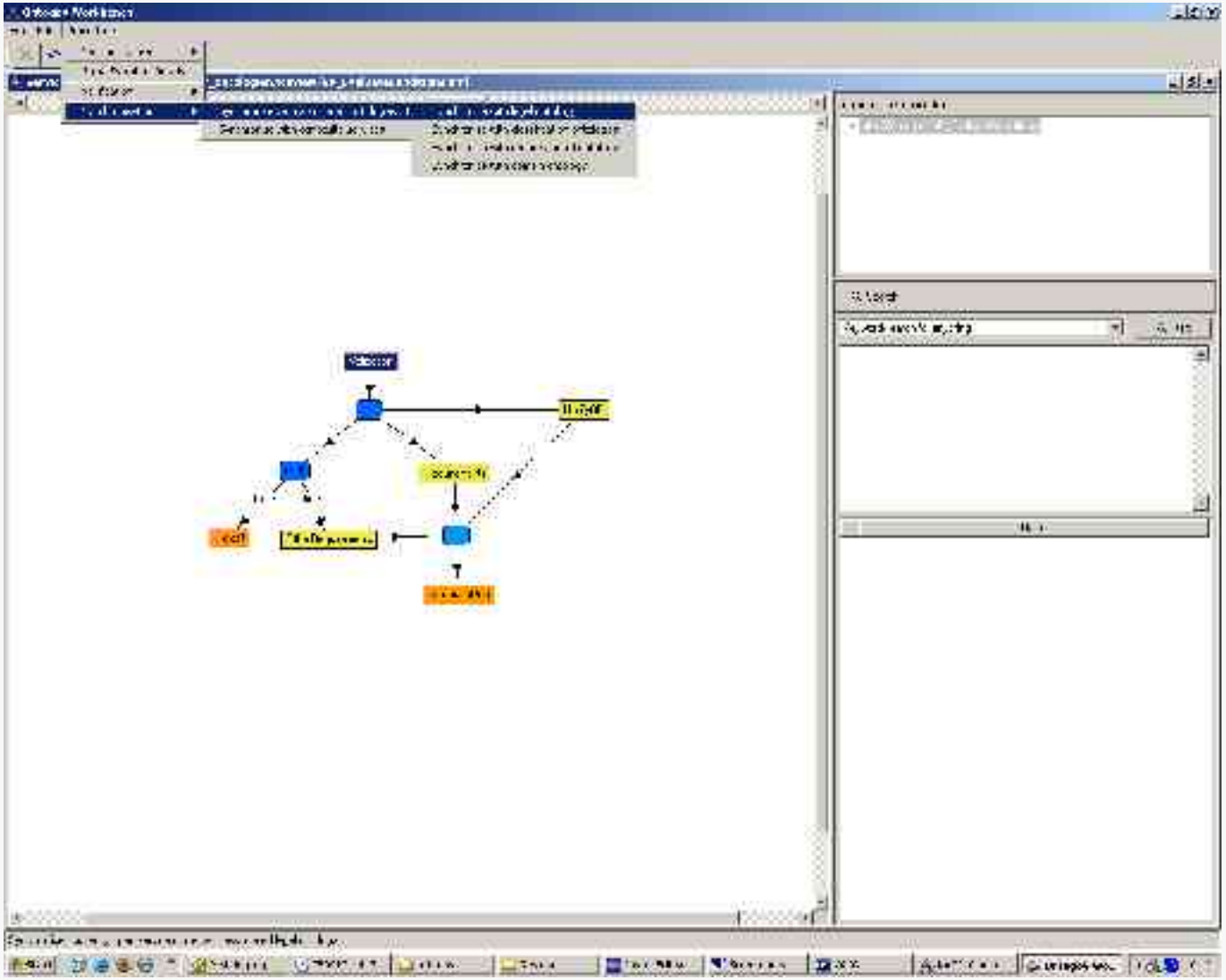


- The role of the change implementation is:
  - to inform a domain expert about all consequences of a change request -**notification**
  - to apply all the (required and derived) changes - **application**
  - to keep track about performed changes – **logging**
    - **Loggin is relateted to** Reversibility which means **undoing all effects of some change**, which may not be the same as simply requesting an inverse change manually

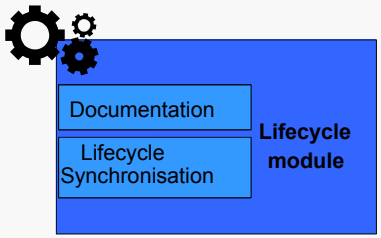
Change propagation



- Two types of change propagation are supported:
  - From the **associated ontologies** to the service description
  - From the **included composite services** to the service description
- The following procedure is realized:
  - The definition of a process model is extended with the **version of each referenced ontology**
  - Changes in the referenced ontologies are **logged in their logs** (which results in the new version of these ontologies)
  - **Pull-based synchronisaton**: On the explicit request all changes between two synchronisatons are discovered
  - Their **impact** on a service model is determined
  - For each „link“ the corresponding **Remove change is recommended**
  - For new entities the **LifeCycle aspects** are taken into account
  - The **domain expert** may accept recommendations or not



Lifecycle

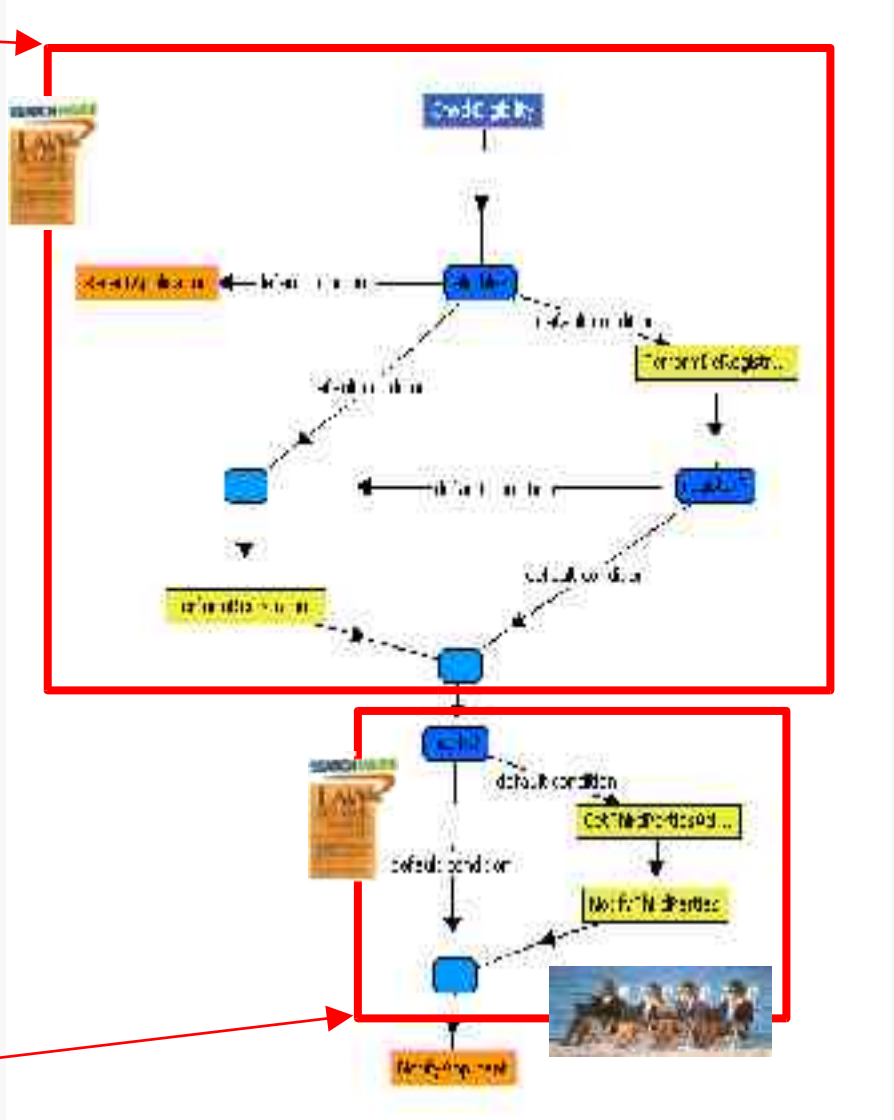


- Describes the knowledge about the modelling of the service model
  - why have two activities been divided
  - why has an activity been added
  - …
- Provides means for describing these decisions and…
  - formally stating reasons for the decisions
  - allowing to find affected decisions, when associated ontologies (e.g. law) are changing

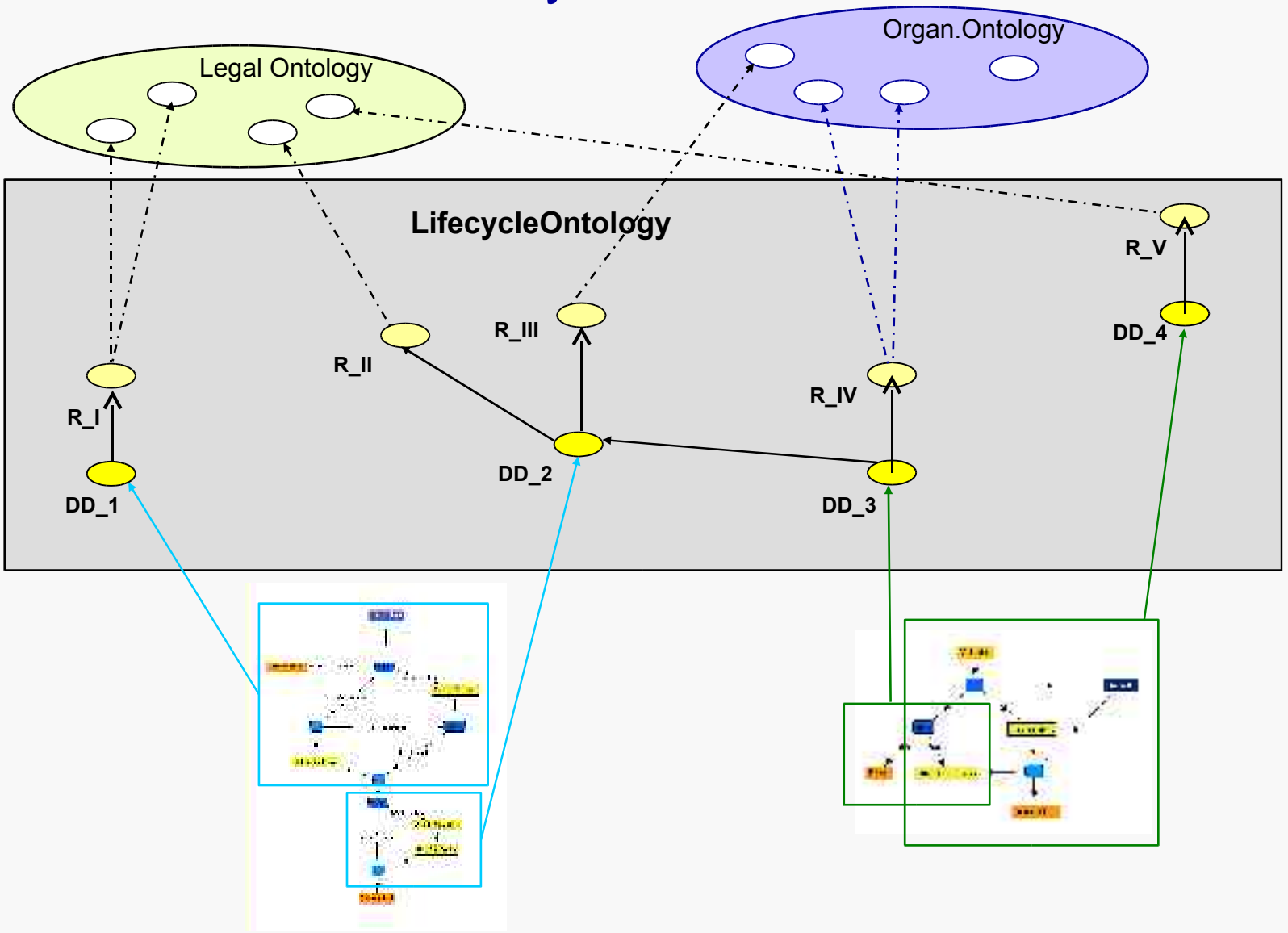
Lifecycle

Design Decision 1:
Eligibility handling
Reason I:
Citizen must have Swiss domicile in order to perform automatic registration/deregistration
Related Instance(s):
SR 101
SR 210 Art. 22A – 26A

Design Decision 2:
Third party notification
Reason II:
Data protection: Transmission of data to third parties
Related Instances:
SR 235 Art. 14
Reason III:
Third Party notification is performed by an external organisation
Related Instances:
BEDAG Information Services



Documentation of Lifecycle



Lifecycle Synchronisation

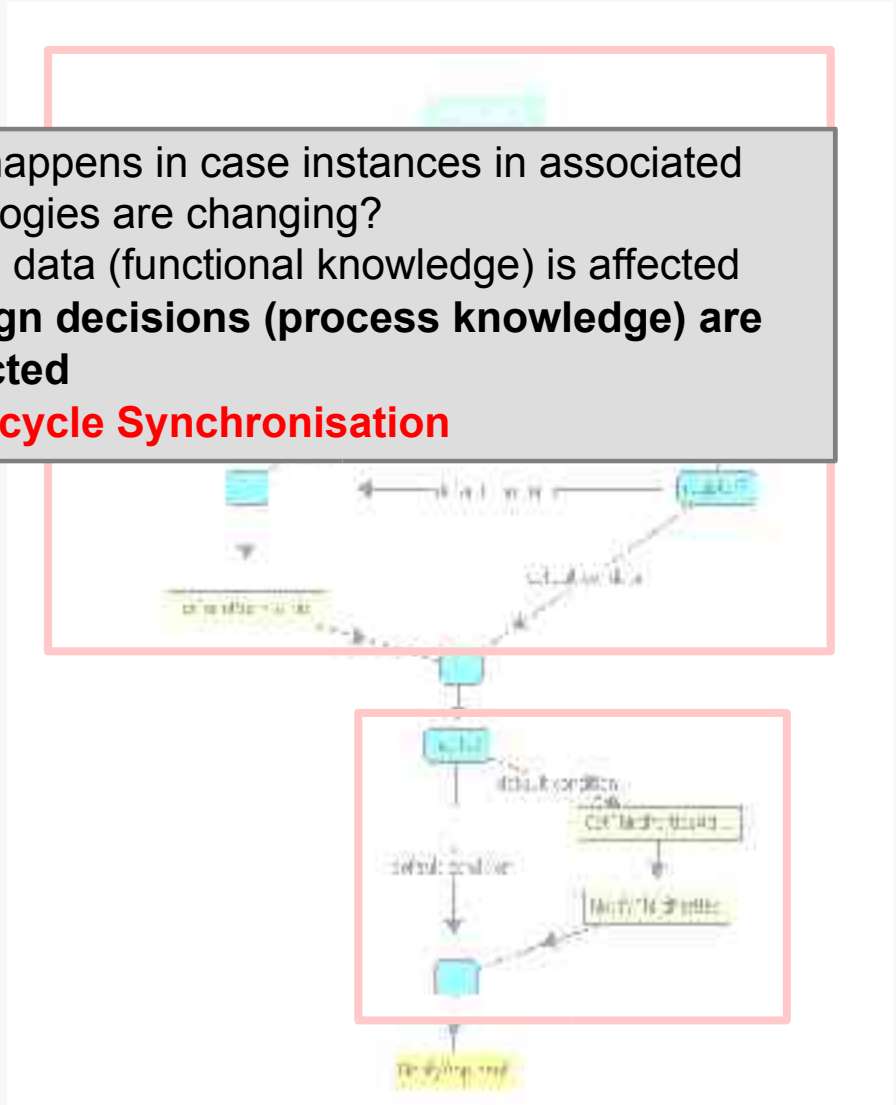
Design Decision 1:
Eligibility handling
Reason I:
Citizen must have Swiss domicile in order to perform registration/deregistration
Related Instance(s):
SR 101
SR 210 Art. 22A – 26A

Design Decision 2:
Third party notification
Reason II:
Data protection: Transmission of data to third parties
Related Instances:
SR 235 Art. 14
Reason III:
Third Party notification is performed by an external organisation
Related Instances:
BEDAG Information Services

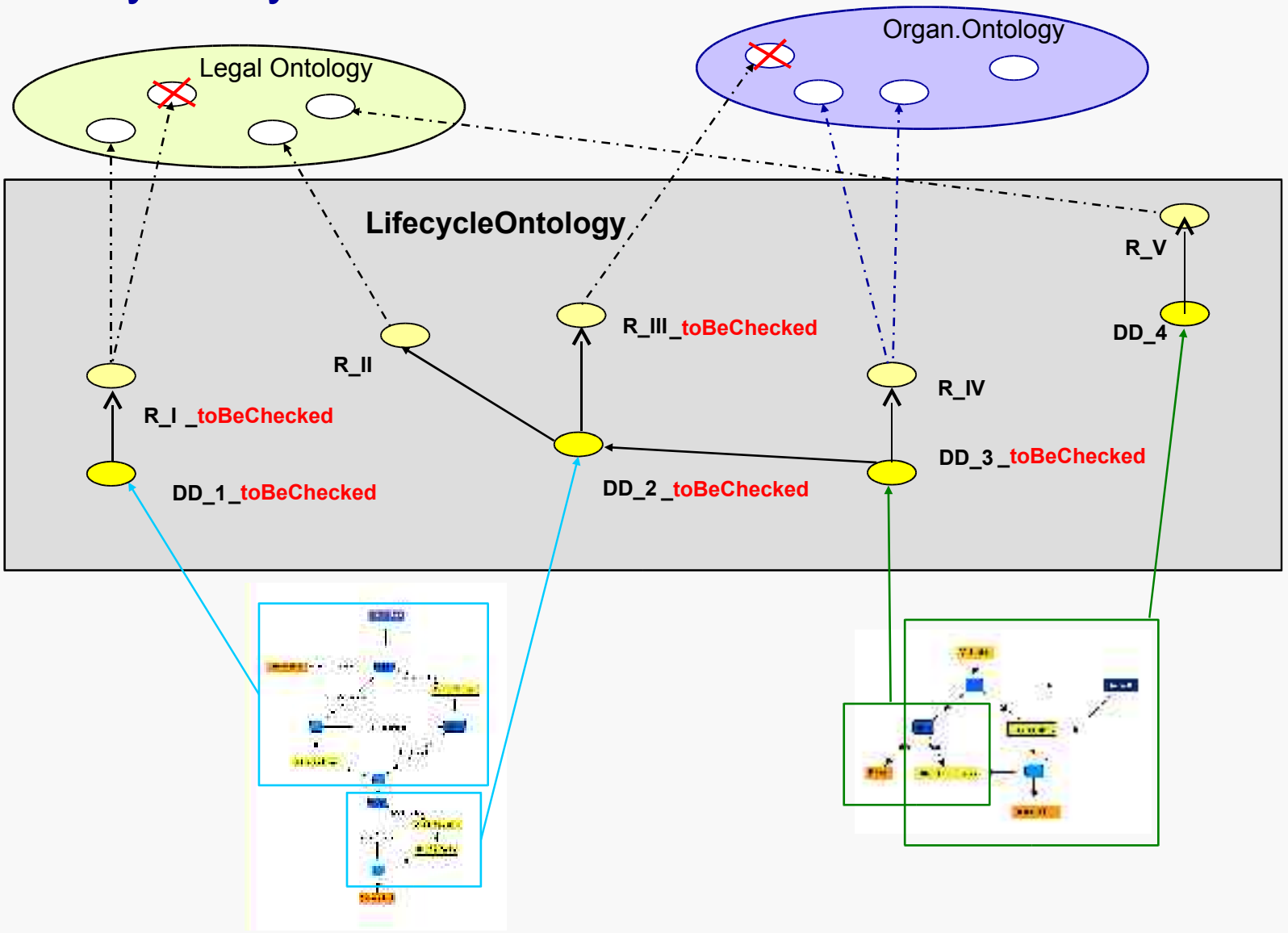
What happens in case instances in associated ontologies are changing?

- meta data (functional knowledge) is affected
- **design decisions (process knowledge) are affected**

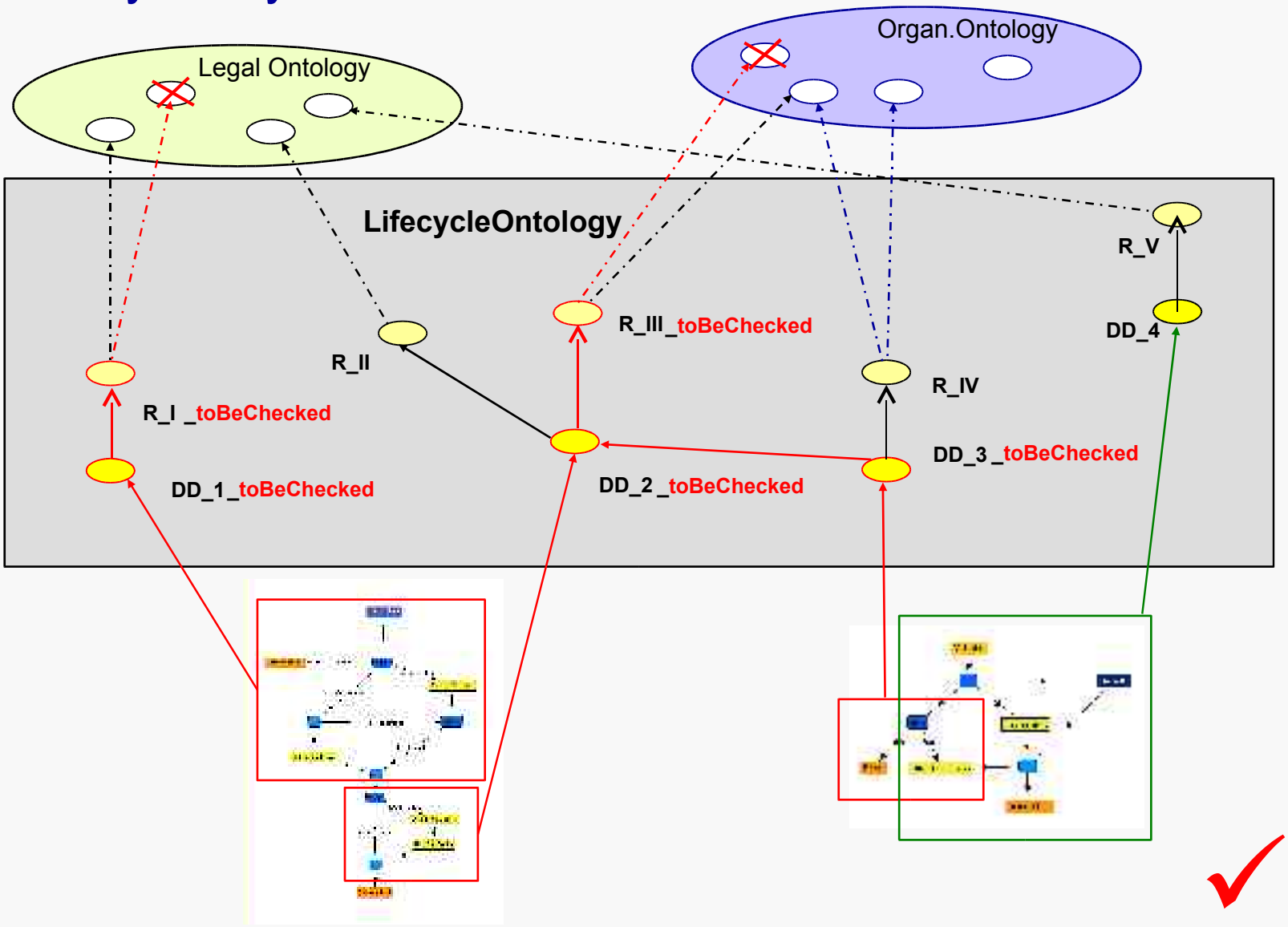
→ **Lifecycle Synchronisation**



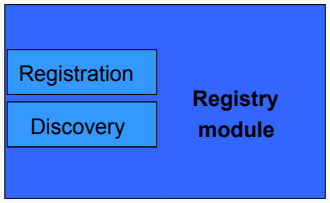
Lifecycle Synchronisation

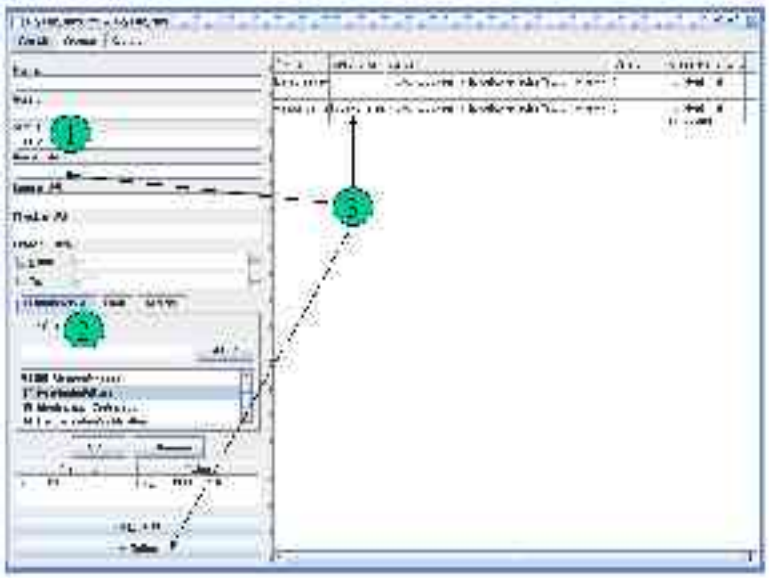


Lifecycle Synchronisation



Registration





- Approach is based on **adding semantic annotations** to service specifications
- This information is used by the registry or by the search engine for providing functionalities such as **searching** and **browsing**
- The main problem is that the domain experts do not have **enough experiences** to describe services
- The **quality of service description** is of vital importance for more accurate and timely decision making

Conclusion

- Ontology-based change management system enables:
  - automatic identification of inconsistencies in the description of the E-Government services (log)
  - analysis of a problem (lifecycle ontology)
  - generation of recommendations for resolving problems

- Advantages:
  - Faster and better service design by all stakeholders involved in the service lifecycle (e.g. managers, software developers)
  - Better control and propagation of changes (e.g. control of changes in law and propagation of changes to the software that delivers the service online)
  - More and better information about each step of the service delivery process, for all stakeholders involved in the service lifecycle

Thanks!

Any questions?

Dimitris Apostolou