Извлечение и интеграция информации из больших данных

Интеграция данных

Брюхов Д.О., Вовченко А.Е., Калиниченко Л.А., Ковалев Д.Ю., Скворцов Н.А.

Институт Проблем Информатики ФИЦ ИУ РАН

План доклада

- Введение
- Процесс извлечения информации из разноструктурированных данных и ее интеграции
- Процесс интеграции данных
  - Поиск релевантных спецификаций
  - Разрешение конфликтов между спецификациями требований и компонентов
  - Конструирование композиций компонентов, уточняющих спецификации требований
- Архитектура средств извлечения информации из разноструктурированных данных, ее интеграции и анализа
- Композиционное проектирование ИС

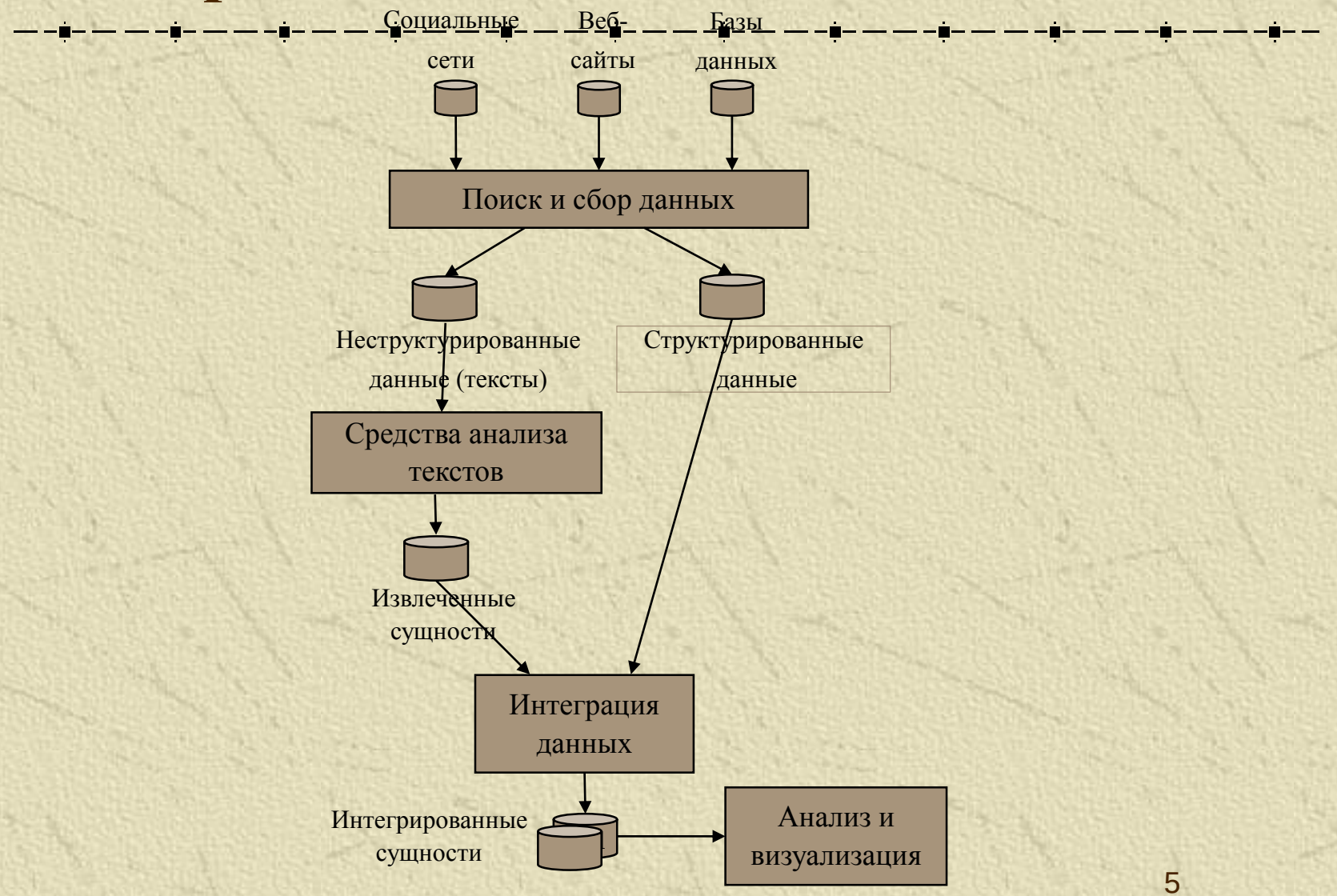
Введение

- Науки с интенсивным использованием данных (НИИД) развиваются в рамках новой парадигмы научных исследований, согласно которой новые знания образуются в результате анализа разнообразных данных, накопленных в результате проведения измерений, наблюдений, моделирования, вычислений.
- Собственно научное знание образуется в процессе интенсивного анализа накапливаемых данных, приводящего в конечном счете к извлечению знаний из данных.
- Объем, разнообразие и качество накапливаемых данных быстро растет.
- Системы с интенсивным использованием данных имеют существенное пересечение с быстро развиваемой областью, именуемой «Big Data».

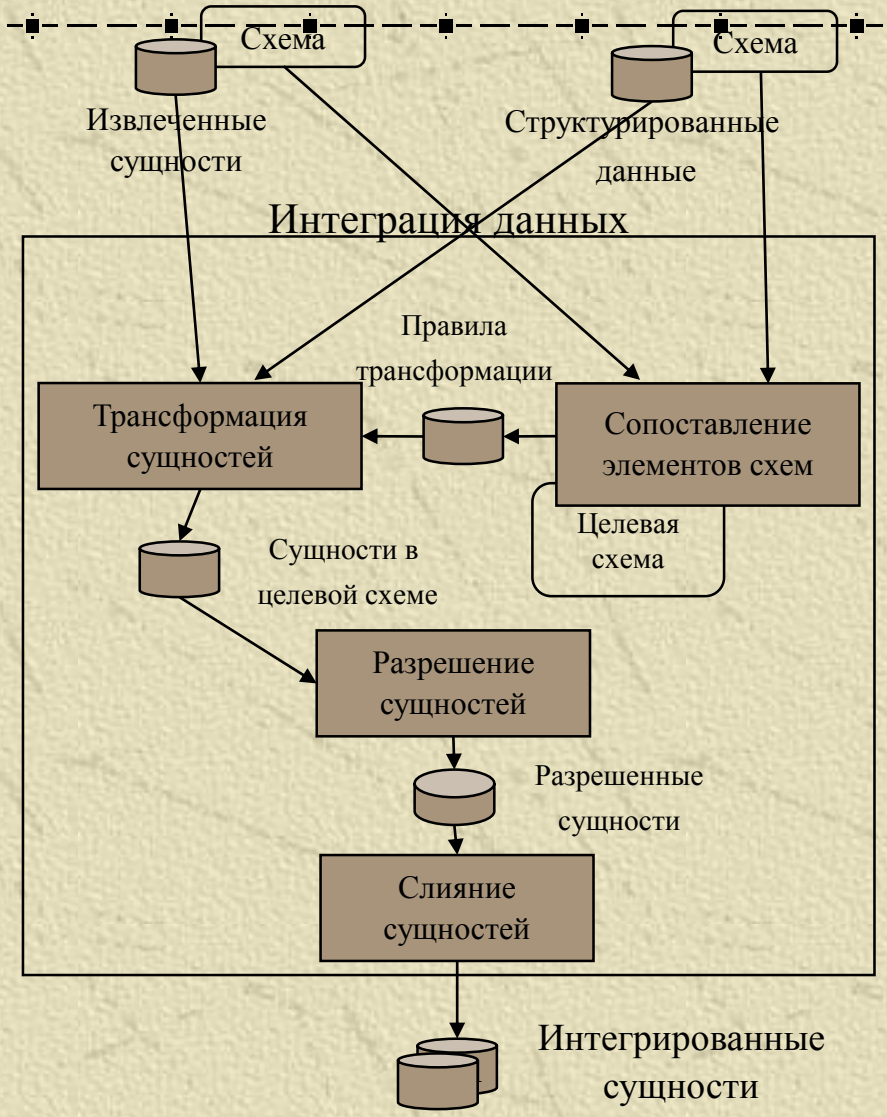
Извлечение структурированной информации

- Данные могут быть структурированными, слабо-структурированными и неструктурированными
- Извлечение структурированной информации из разноструктурированных данных и ее интеграция становится одной из самых важных задач, стоящих перед разработчиками информационных систем.
- Это обусловлено тем, что существующие средства анализа данных, например, анализ многомерных кубов (OLAP) или интеллектуальный анализ данных (data mining), оперируют только структурированной информацией

Процесс извлечения информации из разноструктурированных данных и ее интеграции



Процесс интеграции информации



Процесс интеграции информации. Поиск информационных ресурсов

- Процесс начинается с *поиска информационных ресурсов*, релевантных задаче, и *извлечения* из них *исходных коллекций данных*. Информационные ресурсы могут содержать структурированные данные (базы данных, представленные в различных моделях), слабоструктурированные данные (например, данные из социальных сетей), неструктурированные данные (тексты).

Процесс интеграции информации. Определение схем данных

- На следующем этапе неструктурированные данные (тексты) пропускаются через *средства анализа текстов*, например, Pullenti, Метафраз, AQL, SystemT. При этом из текстов *извлекаются сущности*, например, персоны, организации, территориальные образования и т.д. Сущности, извлекаемые из текстов одним конкретным инструментальным средством, всегда соответствуют одинаковой структуре, определяемой выходным форматом средств анализа текстов. Такая структура называется *исходной схемой данных.*
- Для структурированных данных (извлекаемых из баз данных) исходной схемой является схема базы данных.
- Для слабоструктурированных данных (например, сообщений из социальных сетей) исходная схема образуется путем объединения схемы структурированной части данных (например, профиля пользователя; кем, кому и когда было послано сообщение) и структуры информации, извлеченной из неструктурированной части (текста сообщения).

Процесс интеграции информации. Интеграция информации

- *Сопоставление элементов* исходных схем и целевой схемы может производиться с применением различных методов и средств автоматизации.
- На основании сопоставления элементов схем создаются *правила преобразования данных* из исходных схем в целевую.
- Затем эти правила применяются для *трансформации данных*.
- К коллекциям данных, представленных в единой целевой схеме, применяются более тонкие методы интеграции.
  - Для установления связей между сущностями из различных коллекций применяются методы *разрешения сущностей*, т.е. установления сходства сущностей (в том числе, обнаружение дубликатов).
  - *Слиянием данных* называется образование интегрированного представления информации об одной же сущности реального мира, полученной из различных источников данных. Слияние данных включает слияние кортежей из разных коллекций, соответствующих одной сущности, разрешение возможных конфликтов, обнаружение и удаление ошибочных данных.

Сопоставление схем (schema matching)

- *Сопоставление схем* (schema matching) обычно является первым этапом процесса интеграции данных.
- Эта задача достаточно сложна, поэтому многие системы до сих пор требуют ручной спецификации соответствий элементов схем (с помощью графического интерфейса пользователя).
- Однако, ручная спецификация семантических соответствий между элементами схем слишком трудоемка и подвержена ошибкам в том случае, когда схемы и/или их количество достаточно большие.
- В связи с этим, развиваются методы автоматического или полуавтоматического поиска соответствий.

Подходы к установлении соответствий

- Среди разнообразных алгоритмов установления соответствий между элементами схем, наиболее часто применяемыми являются алгоритмы, основанные на метаданных, которые используют характеристики элементов схем, такие, как имена, типы данных, комментарии.
- Алгоритмы, основанные на экземплярах, устанавливают соответствие между элементами схем на основе близости их экземпляров.
- Для улучшения работы алгоритмов может использоваться дополнительная информация, например, для установления соответствий по именам элементов, они могут быть расширены с помощью общих тезаурусов (таких как Wordnet), или могут использоваться списки и/или тезаурусы синонимов.
- Установление соответствий между элементами схем может быть основано на онтологических спецификациях этих элементов. Элемент спецификации одной схемы онтологически релевантен элементу спецификации другой схемы, если между соответствующими им онтологическими понятиями установлена позитивная ассоциация, или ассоциация обобщения/специализации.
- Установленные соответствия используются для генерации правил преобразования данных из исходной схемы в целевую.

Программные средства для сопоставления схем

- Существует ряд как коммерческих, так и исследовательских программных средств для сопоставления схем.
- В коммерческих средствах, таких, как IBM Infosphere Data Architect, Microsoft Biztalk server, SAP Netweaver Process Integration, или Altova MapForce, сопоставление схем является первым этапом генерации отображений схем (например, для трансформации данных).
- Из некоммерческих средств можно выделить, например, прототип Comma++ [10] для сопоставления схем и онтологий; прототип Harmony [22], являющийся одним из компонентов в проекте Open Information Integration [23] по разработке открытой инфраструктуры для интеграции информации.

Трансформация данных

- *Трансформация данных* представляет собой процесс преобразования данных заданных в исходной схеме в данные представляемые в целевой схеме.
- При этом, целевая схема может быть, как заранее заданной, так и генерируемой на основе интеграции нескольких исходных схем.
- Заданная целевая схема позволяет фиксировать схему под решение конкретных задач, и не зависеть от дальнейшего подключения новых источников данных. Но при этом она может достаточно сильно отличаться от исходных схем, что затрудняет задачу трансформации данных.

Программные средства для трансформации данных

- Для генерации правил трансформации данных используются зависимости между элементами схем, найденные на этапе сопоставления схем. Правила трансформации данных могут задаваться на различных языках трансформации. Обычно средства трансформации данных используют свои собственные языки спецификации трансформаций.
- Одним из первых средств для генерации кода (запросов) для трансформации данных была система Clio [12], разработанная IBM. Clio предоставляет алгоритмы и средства установления соответствий между данными в разных схемах, и автоматической генерации запросов для трансформации данных.
- Проект OpenII (Open Information Integration [23]) предоставляет набор open-source программных средств для интеграции информации, включая трансформацию данных.

Программные средства для трансформации данных

- В настоящее время средства трансформации данных активно используются при создании хранилищ данных (data warehouse). Существует ряд коммерческих средств, таких, как IBM InfoSphere DataStage, Microsoft Integration Services, Oracle Warehouse Builder, Informatica PowerCenter, Clover ETL.
- Существует некоммерческий, open-source вариант продукта Clover ETL [9]. Он включает в себя средство CloverETL Designer для создания и выполнения ETL-графов, и приложения CloverETL Server для администрирования, запуска и мониторинга ETL графов. Clover ETL имеет средства для работы с данными, хранимыми в Hadoop.

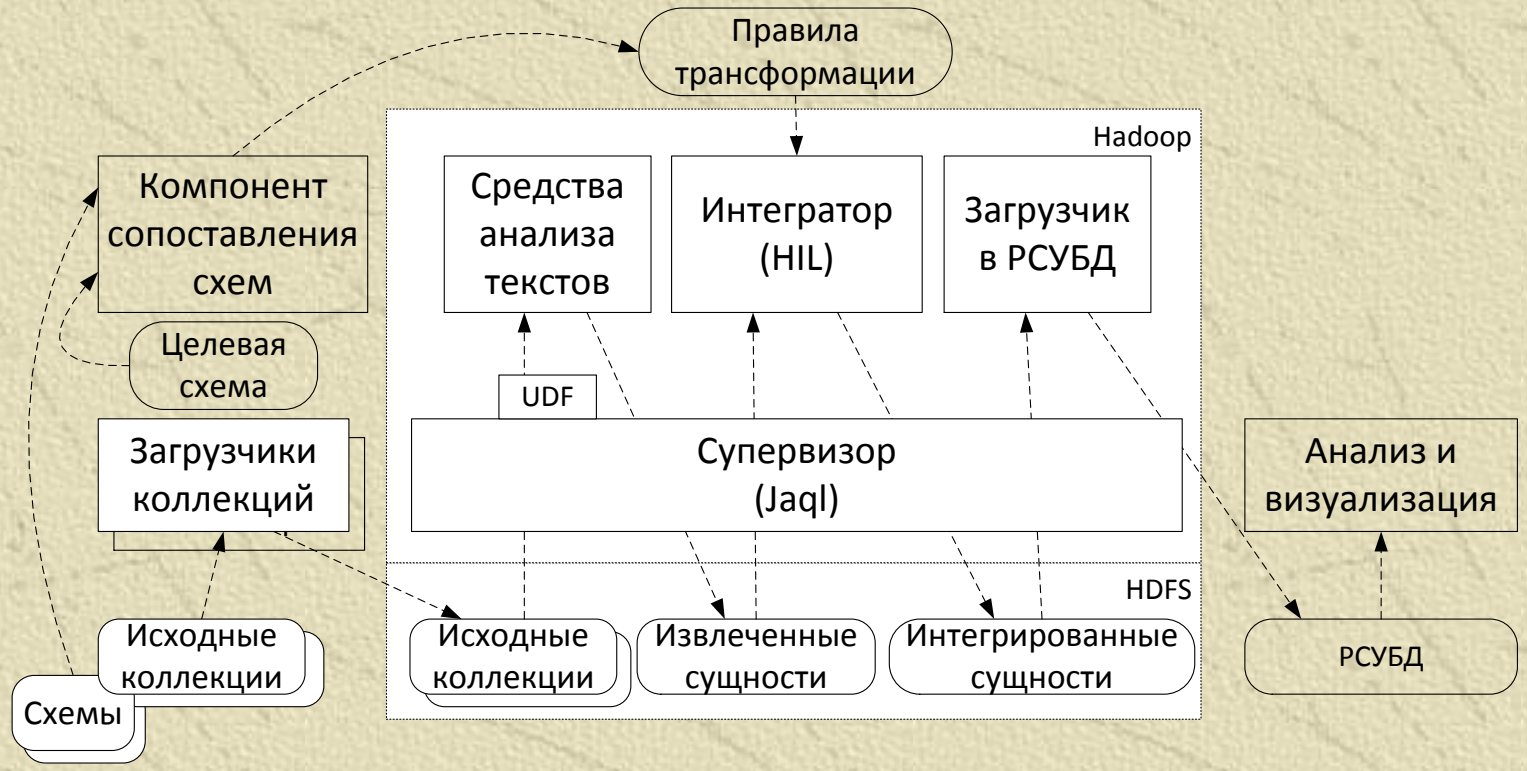
Язык HIL

- Разработанный IBM декларативный язык HIL ориентирован на разрешение и слияние сущностей в Hadoop-инфраструктуре. Язык HIL позволяет реализовать:
  - методы трансформации данных из исходной схемы в целевую. При этом функции разрешения различных конфликтов между этими схемами могут быть описаны также с помощью пользовательских функций (user-defined functions), реализуемых на языках Jaql или Java;
  - методы извлечения, сопоставления и группирования, разбора, связывания, устранения дублирования различных разноструктурированных представлений информации об одних и тех же сущностях реального мира;
  - методы и операции слияния данных об одних и тех же сущностях реального мира и их связей, представленных в разных коллекциях, образованных в процессе разрешения сущностей.

Требования к архитектуре

- Для обеспечения масштабирования рассматриваемого процесса по объему извлекаемых и интегрируемых данных, его необходимо реализовать на основе некоторой платформы распределенного хранения и обработки больших объемов данных.
- В данной работе в качестве такой платформы рассматривается Apache Hadoop.
- Для обеспечения реализации методов интеграции данных над Hadoop (включая разрешение и слияние сущностей), в данной работе используется декларативный язык HIL
- Выбран дистрибутив IBM BigInsights
- В рамках архитектуры предполагается, что коллекции разноструктурированных данных, релевантных задаче, уже отобраны экспертами.
- Также предполагается, что под задачу создана целевая схема (в реляционной модели данных), в которую должны отображаться исходные коллекции.

Архитектура средств извлечения информации и ее интеграции



Композиционное проектирование

- Идея конструирования информационных систем (ИС) из компонентов развивается на протяжении многих лет. По существу, речь идет о мегапрограммировании (Дж.Видерхольд) – крупноблочном конструировании систем.
- Согласно этой идее, целесообразнее осуществлять капиталовложения в создание компонентов, которые можно было бы многократно использовать, чем всякий раз осуществлять разработку ИС сверху – вниз: от спецификаций требований к работающей системе.

Идея композиционного проектирования

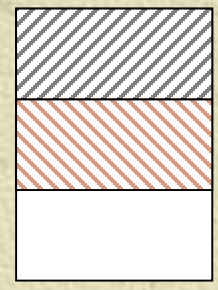
Тип спецификации требований

Тип компонента

Тип компонента

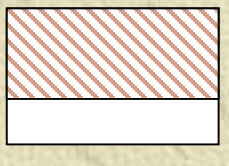
Идея композиционного проектирования

Тип спецификации требований



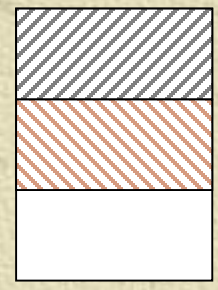
Тип компонента

Тип компонента

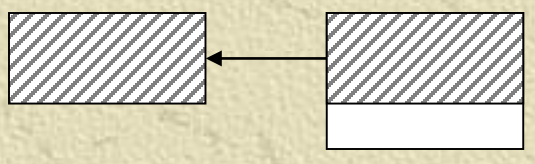


Идея композиционного проектирования

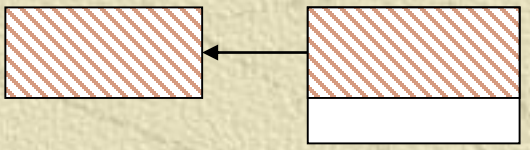
Тип спецификации требований



Тип компонента

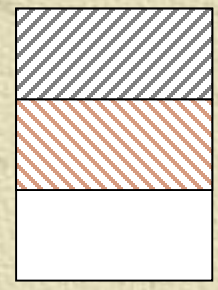


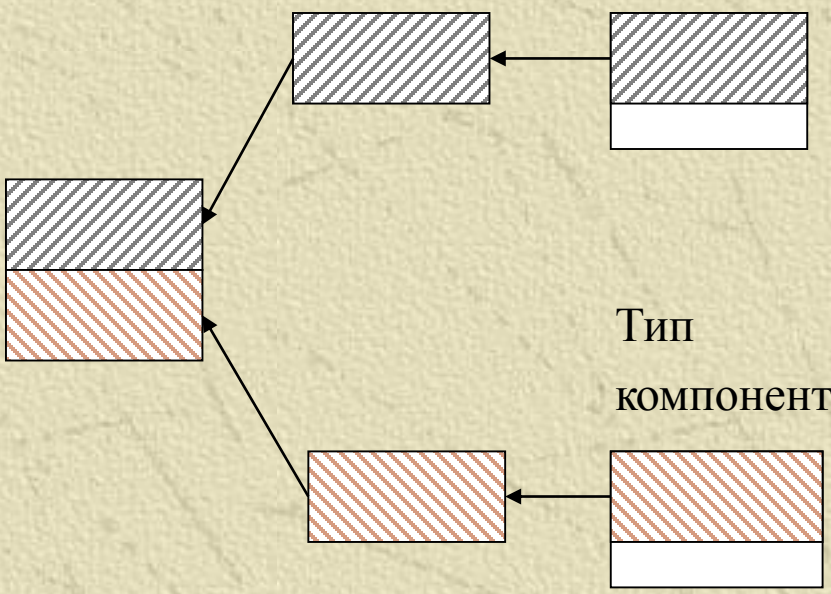
Тип компонента



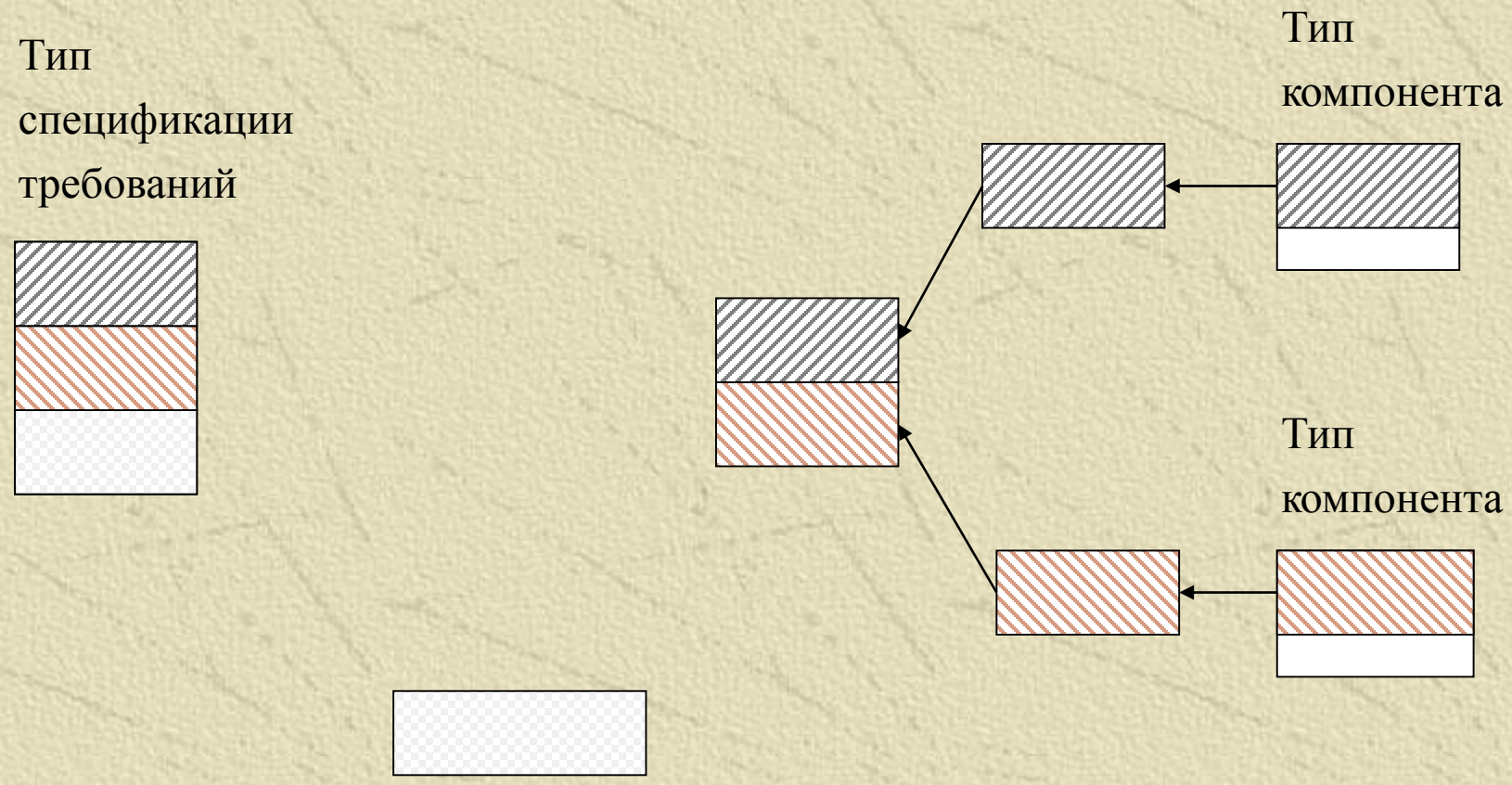
Идея композиционного проектирования

Тип спецификации требований

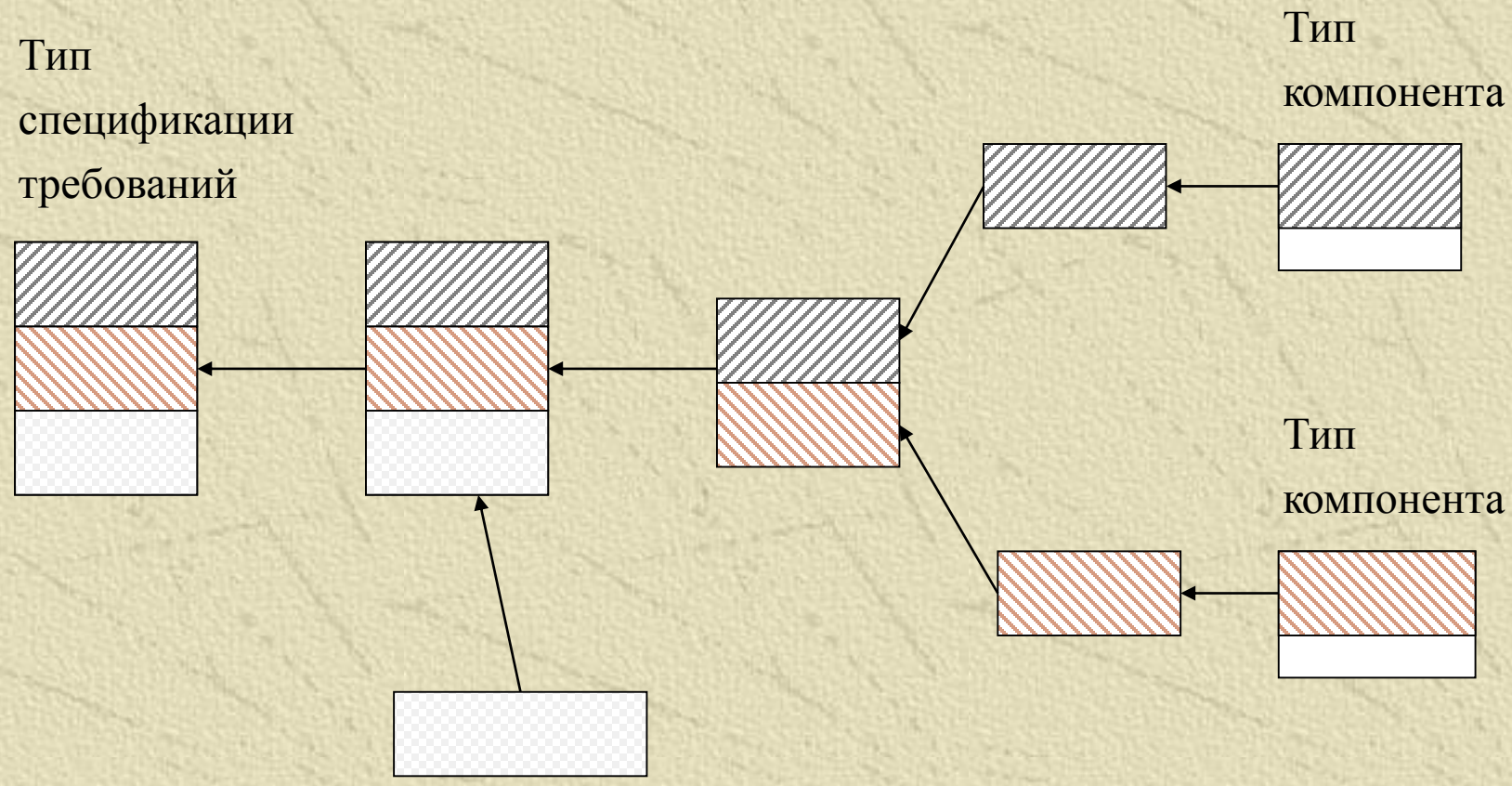




Идея композиционного проектирования



Идея композиционного проектирования



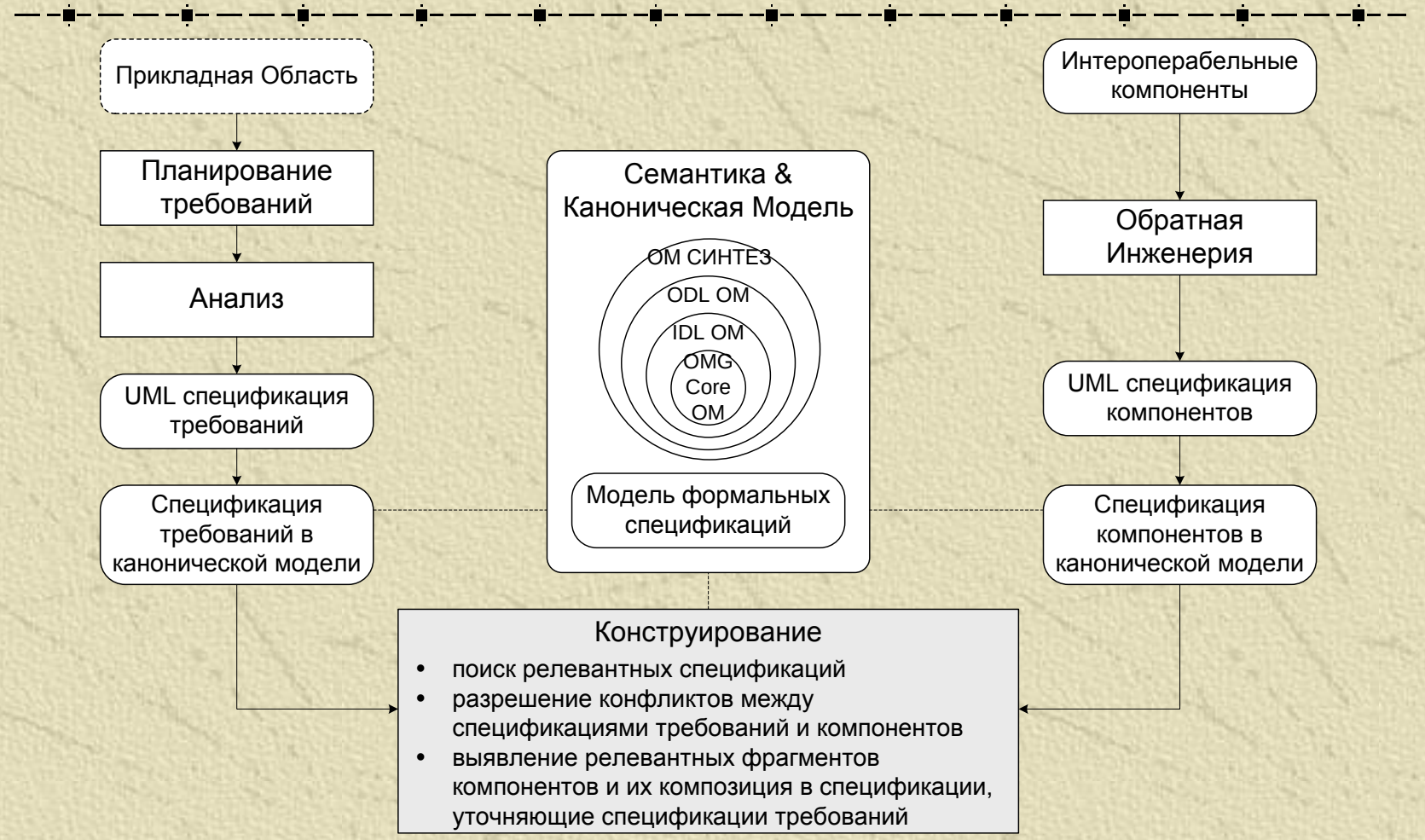
Существующие технологии проектирования ИС из компонентов

- Технологии промежуточного слоя
- Методы и средства компонентно-базированного проектирования ИС
- Методы Объектного Анализа и Проектирования

Основные ограничения существующих технологий

- нет поддержки *семантической интероперабельности*
- не обеспечивается *корректность применения* существующих компонентов в разрабатываемых системах;
- не являются *масштабируемыми* по числу накопленных компонентов;
- отсутствие методов автоматического *разрешения конфликтов* между спецификациями требований и компонентов;
- отсутствие методов *корректной композиции* компонентов.

Схема композиционного подхода к проектированию ИС



Поиск релевантных спецификаций

- При проектировании ИС из готовых компонентов возникает *проблема поиска элементов* (классов, типов, атрибутов, функций) компонентов, подходящих для их использования в разрабатываемой ИС.
- Основой для поиска спецификаций информационных компонентов, семантически релевантных спецификации требований, служит *онтологическая модель*.

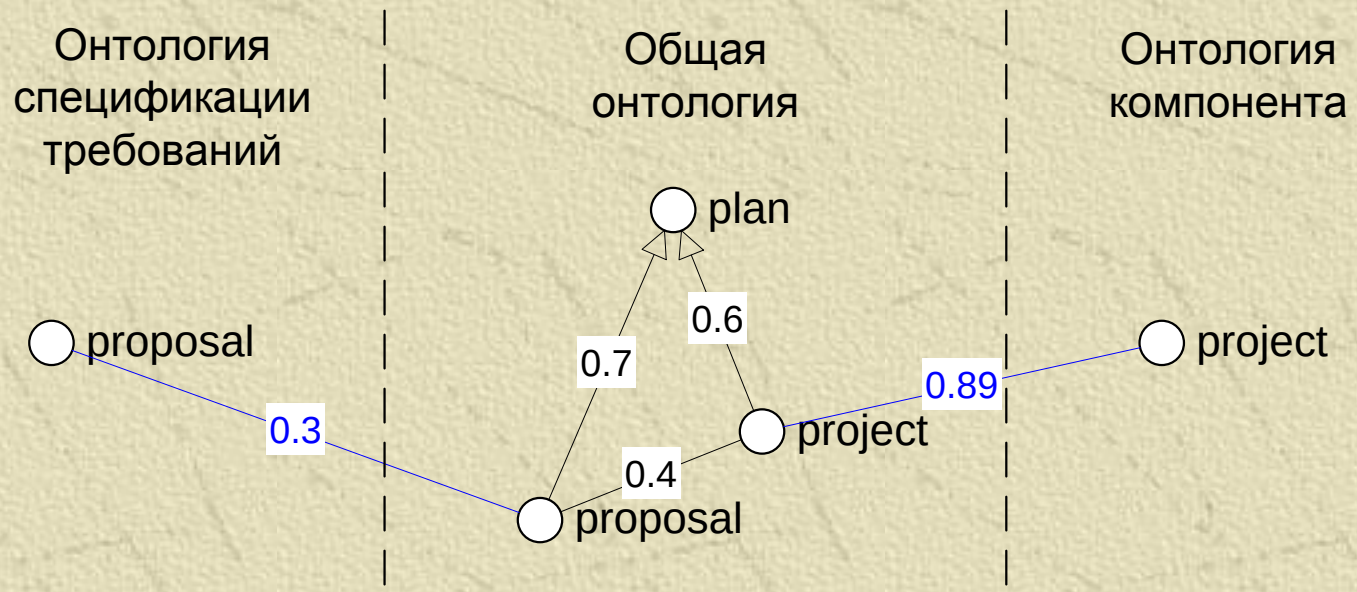
Онтологическая модель

- С каждым элементом спецификации требований и компонентов *связывается* соответствующее онтологическое понятие
- В диссертационной работе спецификация понятия ограничивается его *описанием на естественном языке*. При этом рассматриваются следующие ассоциации между понятиями: *обобщения/специализации* и *позитивные* ассоциации.
- Элемент спецификации компонента *онтологически релевантен* элементу спецификации требований того же вида (класс, тип, атрибут, функция, параметр), если между соответствующими им онтологическими понятиями установлена позитивная ассоциация, или ассоциация обобщения/специализации.

Пример спецификации онтологий

```
{proposal;
  in: concept;
  positive: {{project, 0.4}};
  def: {“a research project proposal or submission
         presented by an organization or group of
         organizations for evaluation”}
}
{project;
  in: concept;
  positive: {{proposal, 0.4}, {research, 0.6}};
  def: {“a scientific or study research accepted
         to be realized”}
}
```

Отображение понятий в общую онтологию

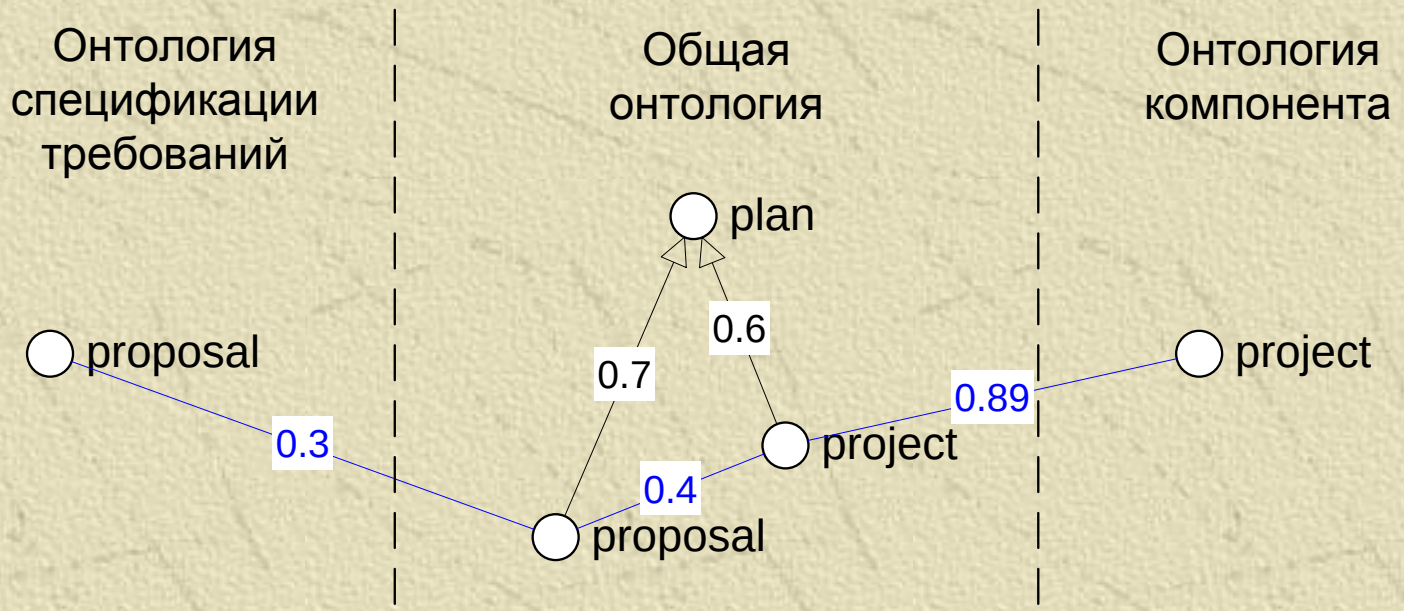


Функции корреляции

- Функция позитивной корреляции
  - $sim(X,Y) = (Cx,Cy) / |Cx|*|Cy|$

- Функции вычисления ассоциаций обобщения/специализации
  - $r(X,Y) = \Sigma_i(min(Cx_i,Cy_i)) / |Cx|$
  - $r(Y,X) = \Sigma_i(min(Cx_i,Cy_i)) / |Cy|$

Композиция понятий



WEIGHT(proposal, project) = (0.3 τ_2 0.4 τ_2 0.89) = 0.11

Композиция ассоциаций

- Два понятия являются *связанными*, если существует *путь* от одного понятия в другое.

	p	g	s
p	p	p	p
g	p	g	p
s	p	p	s

Вид пути

	p	g	s
p	τ_2	τ_2	τ_1
g	τ_2	τ_3	τ_1
s	τ_1	τ_1	τ_3

Коэффициент корреляции пути

- t-нормы для пересечения нечетких множеств
  - $\tau_1(a,b) = \max(0, a+b-1)$
  - $\tau_2(a,b) = ab$
  - $\tau_3(a,b) = \min(a, b)$

Алгоритмы поиска онтологически релевантных элементов

- алгоритмы отображения понятий спецификации требований (и компонентов) в понятия общей онтологии;
- алгоритмы установления ассоциаций между понятиями спецификации требований и компонентов на основе композиции ассоциаций между понятиями;
- алгоритмы установления ассоциаций между элементами схем спецификации требований и компонентов.

Разрешение конфликтов

- При проектировании ИС из готовых компонентов неизбежно возникают различные конфликты между спецификациями требований и компонентов.
- Конфликты могут возникать как из-за разных областей применения, так и из-за разного видения разработчиками представления одной и той же задачи.

Виды конфликтов

- Структурные конфликты
  - конфликт имен
  - использование различных типов атрибутов
  - использование разных уровней абстракции представления информации
- Конфликты значений
  - использование разных отношений на экстенсионалах классов
  - представление атрибутов в различных единицах измерения
- Конфликты поведения
  - использование различных утверждений и ограничений целостности
  - использование различных реализаций функций

Подходы к разрешению конфликтов

- применение языка высокого уровня для описания функций разрешения конфликтов
  - Функции разрешения конфликтов задаются с помощью формул языка СИНТЕЗ.
  - Язык формул является вариантом типизированного языка логики первого порядка.
- применение набора предопределенных правил структурных преобразований
  - Для разрешения конфликтов структурного вида используются предопределенные правила.
  - Они устанавливают релевантность путей спецификаций требований и компонентов, и задают правила построения функций разрешения конфликтов.

Пример описания преобразований на языке СИНТЕЗ

Имеем 2 релевантных типа *Proposal* и *Submission* с разным набором атрибутов:

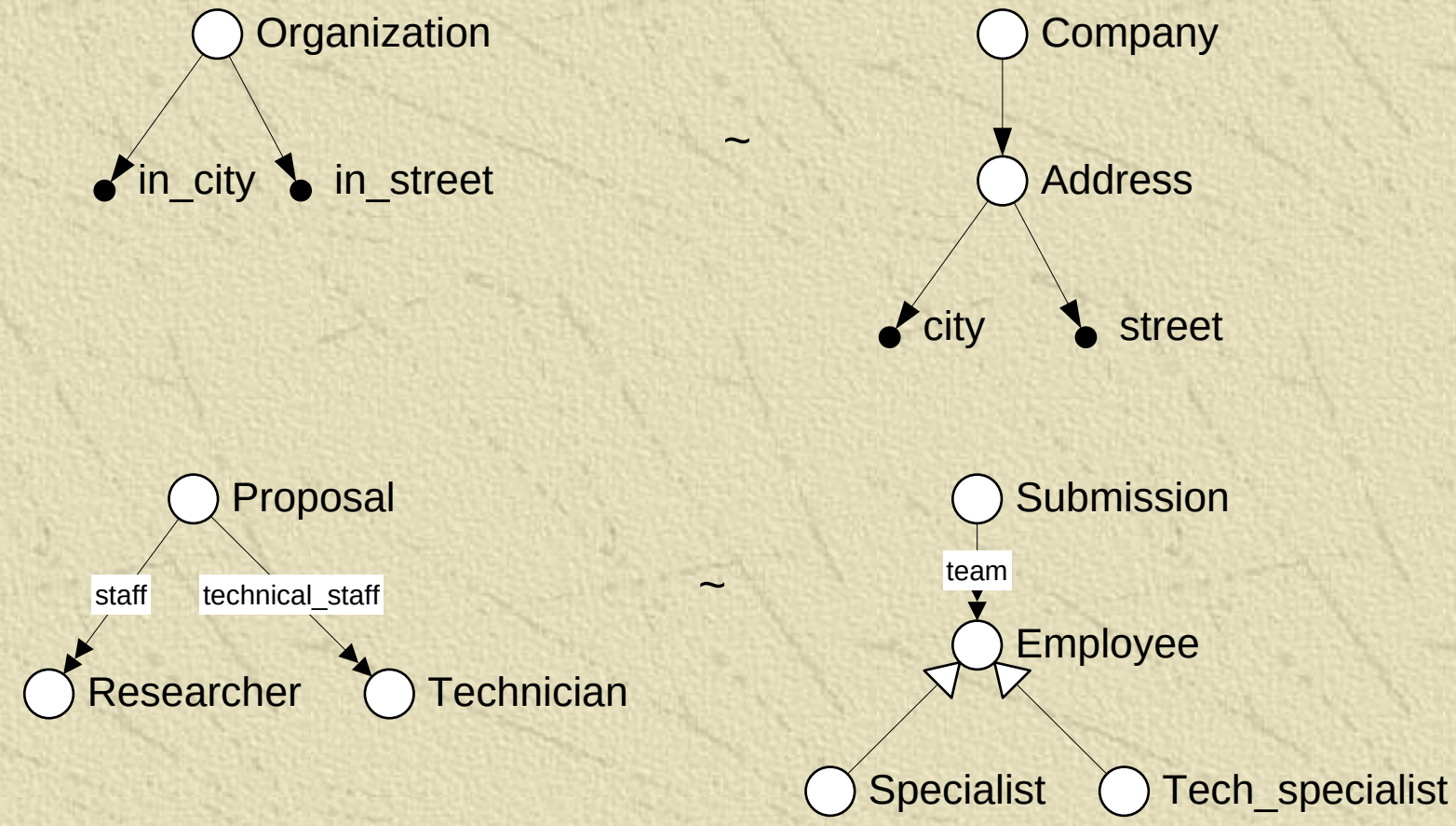
```
{Proposal;
  in: type;
  starting_date: time;
  termination_date: time;
}
```

```
{Submission;
  in: type;
  starting_date: time;
  duration: time;
}
```

Функция разрешения конфликта для атрибута *termination_date* выглядит следующим образом:

```
get_termination_date: {in: function;
  params: {+s/Submission, -returns/time};
  {{ returns = s.starting_date + s.duration }}
}
```

Структурные конфликты



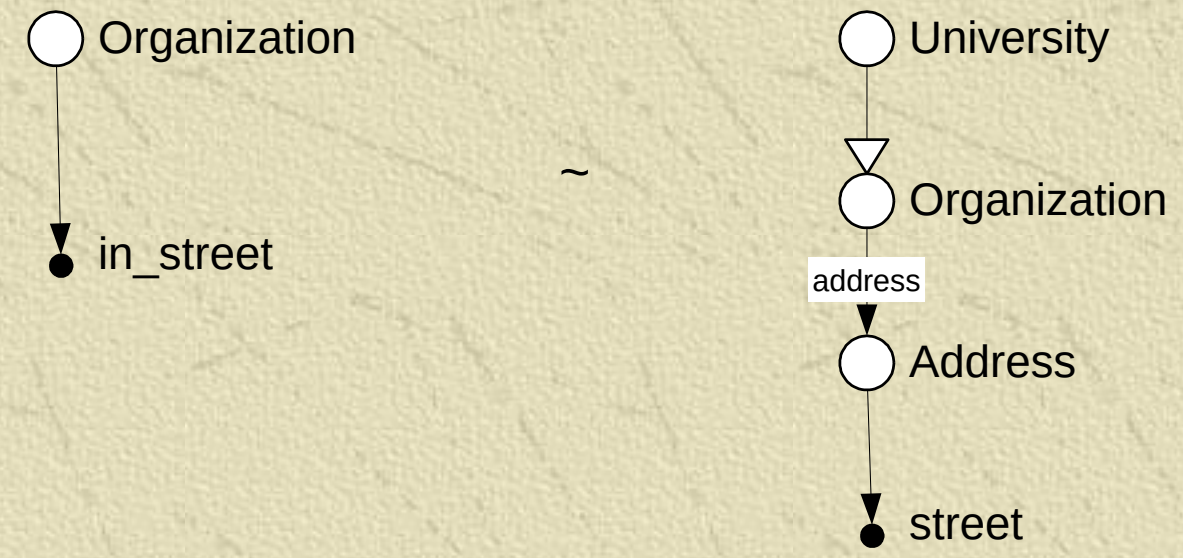
Подход к разрешению структурных конфликтов

- Путь *X1...X2* является *релевантным* пути *Y1...Y2*, если они удовлетворяют одному из восьми правил структурных преобразований, введенных в диссертации.
- Условия потенциальной релевантности путей
  - полнота
  - ацикличность
  - минимальность

Пример правила структурных преобразований

- Путь *X0...X1* релевантен пути *Y0...Y1*, если они удовлетворяют условию потенциальной релевантности, описанному выше, и имеют следующий вид:
  - $X0–a1 \rightarrow X1 \sim Y0(–b \rightarrow | \Rightarrow)^*Y3–b1 \rightarrow Y1$
  - $X0 \rightarrow X1 \sim Y0(–b \rightarrow | \Rightarrow)^*Y3 \rightarrow Y1$
- При этом функции разрешения конфликтов имеют следующий вид:
  - ```
    get_a1: {in: function;
      params: {+y/Y0, -returns/X1};
      {{ returns = y(.b)*.b1 }}}
    ```
  - ```
    get_X1: {in: function;
      params: {+y/Y0, -returns/X0.X1};
      {{ returns = y(.b)*.Y1 }}}
    ```

Пример применения правила структурных преобразований




```
get_in_street: {in: function;
  params: {+y/University,
           -returns/Organization.in_street};
  {{ returns = y.address.street }}
}
```

Алгоритм поиска релевантных путей

- Для произвольной пары типов – типа спецификации требований Ts_1 и типа компонента Tr_1 из списка онтологически релевантных типов:
  - находится тип спецификации требований Ts_2 и тип компонента Tr_2 из списка онтологически релевантных типов, такие, что пути $Ts_1 \ldots Ts_2$ и $Tr_1 \ldots Tr_2$ удовлетворяют одному из правил структурных преобразований;
  - если эксперт подтверждает релевантность найденных путей, они заносятся в список релевантных путей.

Конструирование композиций компонентов

- Процесс конструирования основан на *выявлении фрагментов спецификаций* существующих компонентов и их дальнейшей *композиции*, уточняющей спецификацию требований.
- Для этого используются операции над типами, ведущими к трансформации их спецификаций – операции *декомпозиции* и *композиции*.
- *Уточняющие спецификации*, образуемые при конструировании, согласно теории уточнения, могут использоваться всюду *вместо уточняемых* спецификаций требований, так что пользователи не замечают этой замены.

Общий редукт

- *Редукт* R_T типа *T* определяется как подсигнатура сигнатуры типа, при этом множество атрибутов редукта является подмножеством множества атрибутов типа, множество функций редукта является подмножеством множества функций типа, множество предикатов редукта является подмножеством множества предикатов типа.
- *Общий редукт* для типов *T1*, *T2* есть редукт R_{T1} типа *T1* такой, что существует редукт R_{T2} типа *T2*, при этом редукт R_{T2} является уточнением редукта R_{T1}. Редукт R_{T2} называется сопряженным редуктом к общему редукту.

Алгоритм конструирования наиболее общего редукта

- Для определения наиболее общих редуктов для каждой пары онтологически релевантных типов *Ts* и *Tr* требуется найти максимальный набор *A* пар атрибутов (a_{Ts}, a_{Tr}), которые являются онтологически релевантными и имеют типы такие, что атрибут a_{Tr} можно использовать вместо a_{Ts}. В *A* включаются:
  - атрибут a_{Ts} и атрибут a_{Tr}, если тип атрибута a_{Ts} является супертипом типа атрибута a_{Tr}
  - функция f_{Ts} и функция f_{Tr}, если они имеют эквивалентные сигнатуры с точностью до отношения на типах параметров (контравариантность для входных параметров и ковариантность для выходных параметров)
  - атрибут a_{Ts} и функция f_{Tr}, если f_{Tr} является функцией разрешения конфликтов для атрибута a_{Ts}

Пример спецификации редукта и конкретизирующего редукта

```
{R_Prop_Subm;
  in: reduct;
    metaslot
      of: Proposal;
      taking: {name, area, consortium, budget,
               starting_date, termination_date};
      c_reduct: CR_Prop_Subm;
    end
}
{CR_Prop_Subm;
  in: c_reduct;
    metaslot
    of: Submission;
    taking: {name, research_area, participates,
             req_money, starting_date, duration};
    reduct: R_Prop_Subm;
    end;
```

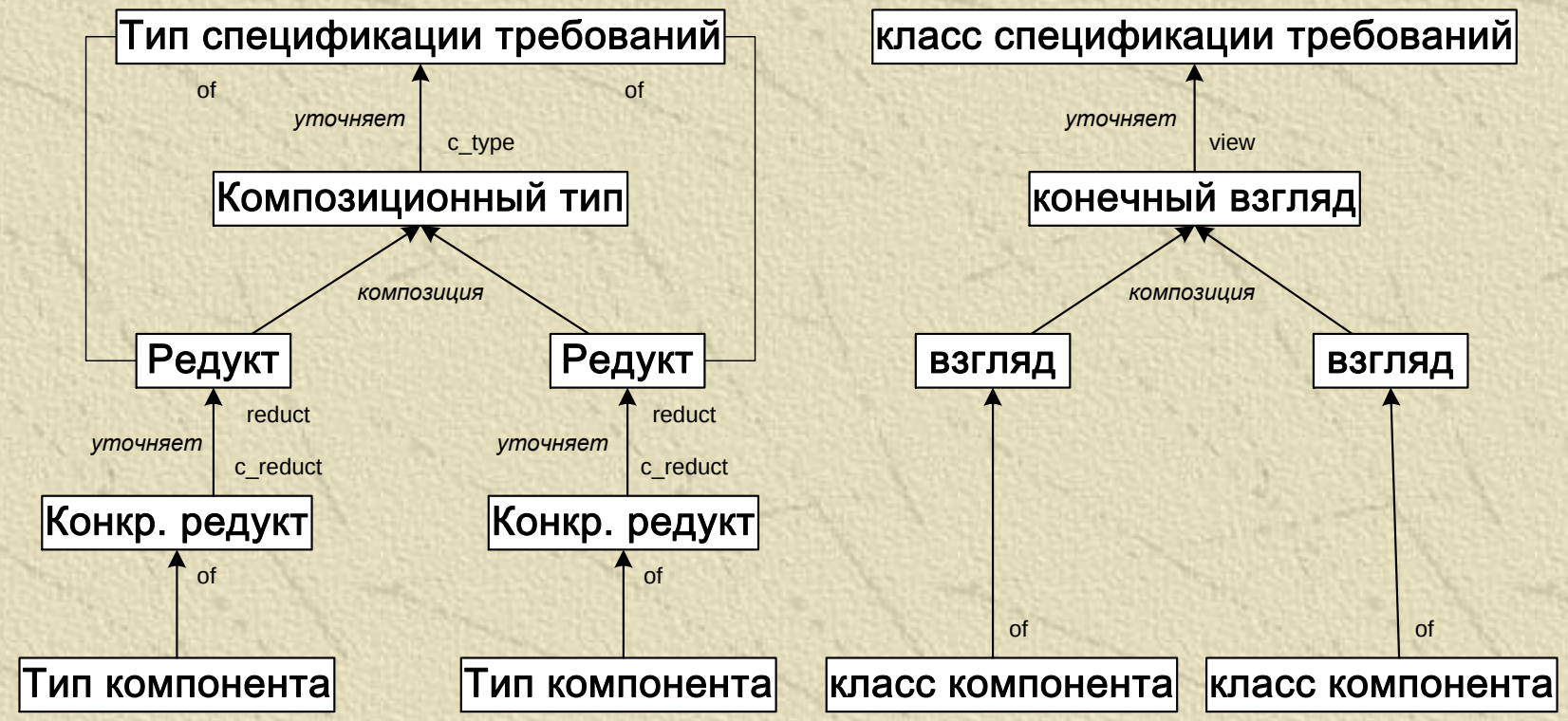
Пример спецификации редукта и конкретизирующего редукта (2)

```
  simulating: {
    p.name              ~ s.name,
    p.area              ~ s.research_area,
    p.consortium ~        s.participates,
    p.budget            ~ s.req_money,
    p.starting_date     ~ s.starting_date,
    p.termination_date ~ f_termination_date
  };
  f_termination_date: {in: function;
    params: {+ext/CR_Prop_Subm, -returns/time};
    predicative: {ex s/Submission (s =
ext/CR_Prop_Subm) &
        returns = s.starting_date+s.duration }}}
}
```

Операции композиции спецификаций

- Операции композиции типов
  - meet
  - join
- Операции композиции классов
  - объединение
  - пересечение
  - соединение

Иерархия спецификаций композиционных типов и классов



Алгоритм конструирования уточняющих спецификаций

- При реализации программных систем задаются только спецификации типов. Композиция редуктов строится таким образом, чтобы покрыть как можно больше атрибутов типа спецификации требований.
- Если при этом некоторые атрибуты остаются не поддержанными, создается новый тип, содержащий такие атрибуты.
- В дальнейшем, разработчик должен реализовать этот тип.
- Композиция редуктов осуществляется посредством операции *join*.

Алгоритм конструирования уточняющих спецификаций (2)

- При разработке информационных систем главными являются классы. В зависимости от требований при композиции используются операции объединения, пересечения или соединения классов.
- При объединении классов тип экземпляра конструируемого взгляда образуется посредством операции *meet*, применяемой к типам экземпляров классов-операндов.
- При пересечении (соединении) классов тип экземпляра конструируемого взгляда получается посредством операции *join*, применяемой к типам экземпляров классов-операндов.

Пример спецификации взгляда

```
{<v_prop_subm_res>;
    in: class;
    metaslot
      view_gen:{ in: function;
        params: {-returns/v_prop_subm_res as_class};
        enforcement: on_access;
        {{ v_prop_subm(s) | v_prop_res(r) }}}
    end;
    instance_section: CT_Prop_Subm_Res;
    class_section: {
        key: invariant, {unique; {name}};
    }
}
```

```
CT_Prop_Subm_Res = R_Prop_Subm ⊓ R_Prop_Res
```

Synthesis - OntologyIntegration

Project Phase Help

0.1 Set Edit word forms

AOM & ROM Concepts

funding_agency

a group of persons working together; company

company
group
organize
person
together

Add Edit Remove

Diagram Synthesis

Find

{ O_Organization;
in: concept;
def: { a group of persons working together; company };
}

APP funding_agency
Proposal
Organization

COM Concepts

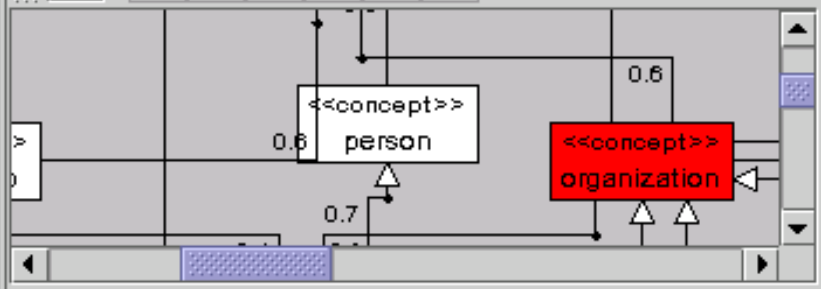
COM

a group of persons organized for some work; company

company
group
organize
person
work

Diagram Synthesis

Find



organization (0.8352952)
workgroup (0.59705174)
employee (0.4504046)
staff (0.2699581)
association (0.2127884)

0.8352952 Change Accept Reject Reject others

Synthesis - OntologyIntegration

Project Phase Help

Application

funding_agency

an area, a subject of the proposal

<<type>>
Proposal
name:string
area:string
budget:real
status:string
starting_date:time
termination_date:time
expmax:integer
expnumb:integer
checked:boolean
accept_proposal(m,t)

offer

participate_at

Proposal
name
area
budget
status

Resource

university_grants

a subject of the proposal

<<type>>
University_research
name:string
area:string
status:string
req_funding:integer
starting_date:time
termination_date:time
rev_max:integer
rev_numb:integer
examined:boolean
accept_research(st,fin)

offer

projects

project_managed

University_research.name (0.13183482)
Organization.branch (0.13059528)
University_research.area (0.11944841)
Review.located (0.10764267)
Professor.works (0.1010568)

Accept Reject Reject others

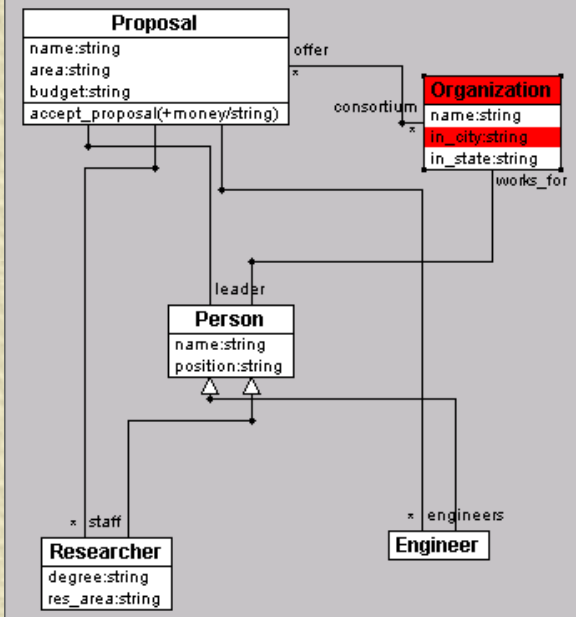
Synthesis Design Method Prototype: Path Phase

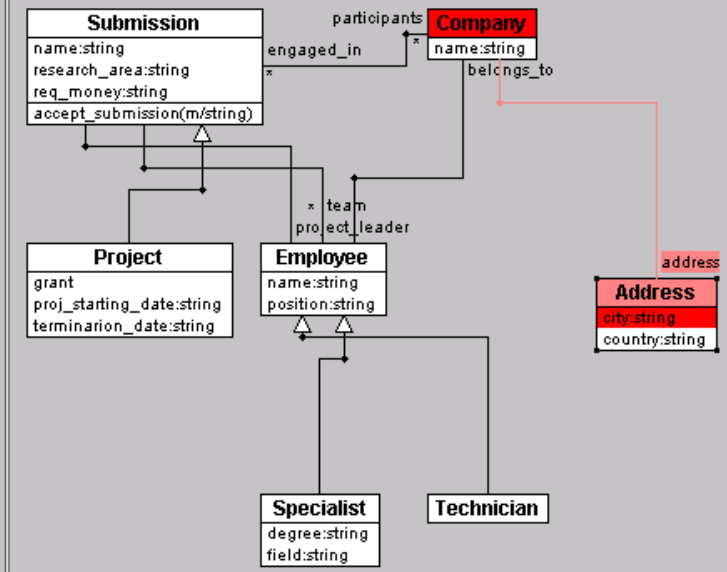
next phase

funding_agency

industrial_labs

city where the organization is located





Organization.name
Organization.offer
Organization.in_city
Organization.in_state

? Company.address.Address.city

accept path conformance

reject path conformance

Synthesis Design Method Prototype: Reduct Phase

next phase

funding_agency

a name of the proposal

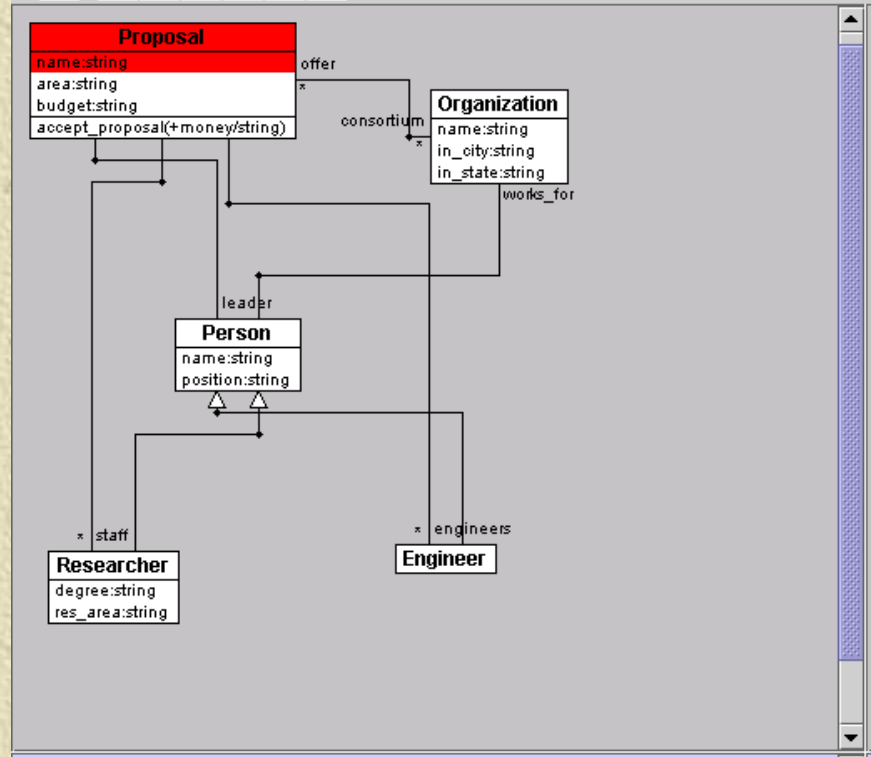
```
{ R_Proposal_Submission;
  in: reduct;
  metaslot
    of: Proposal;
    c_reduct: CR_Proposal_Submission;
    taking: {
      name ,
```

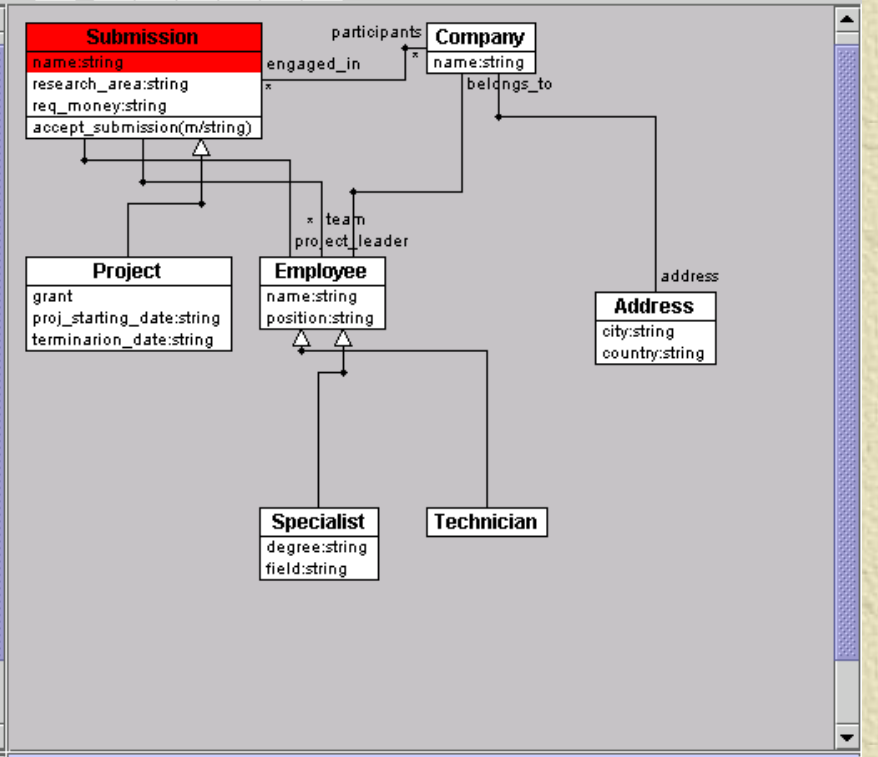
industrial_labs

name of the submission

```
reduct: R_Proposal_Submission;
  simulating: {
    all s/Proposal ex r/Submission(
      s.name  = r. name &
      s.area  = r. research_area &
      s.budget = r. req_money &
```

downButtonIcon





Proposal.name
Proposal.area
Proposal.budget

+ Submission.name

accept reducts

reject type relevance

Synthesis Design Method Prototype: Concretizing Type Phase

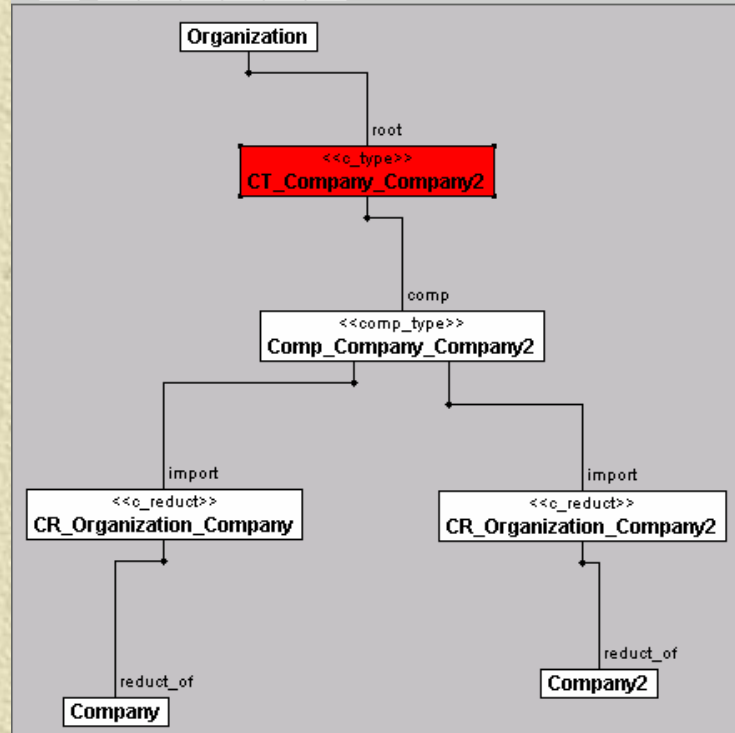
next phase

funding_agency

industrial_labs

```
composing: {{
Comp_Company_Company2 = CR_Organization_Company[name, offer, in_city, in_state] | CR_Organization_Company2[name, offer, in_city, in_state]
}}
simulating: {
  all s/Organization ex r/Comp_Company_Company2(
      s.name = r.
```

name &



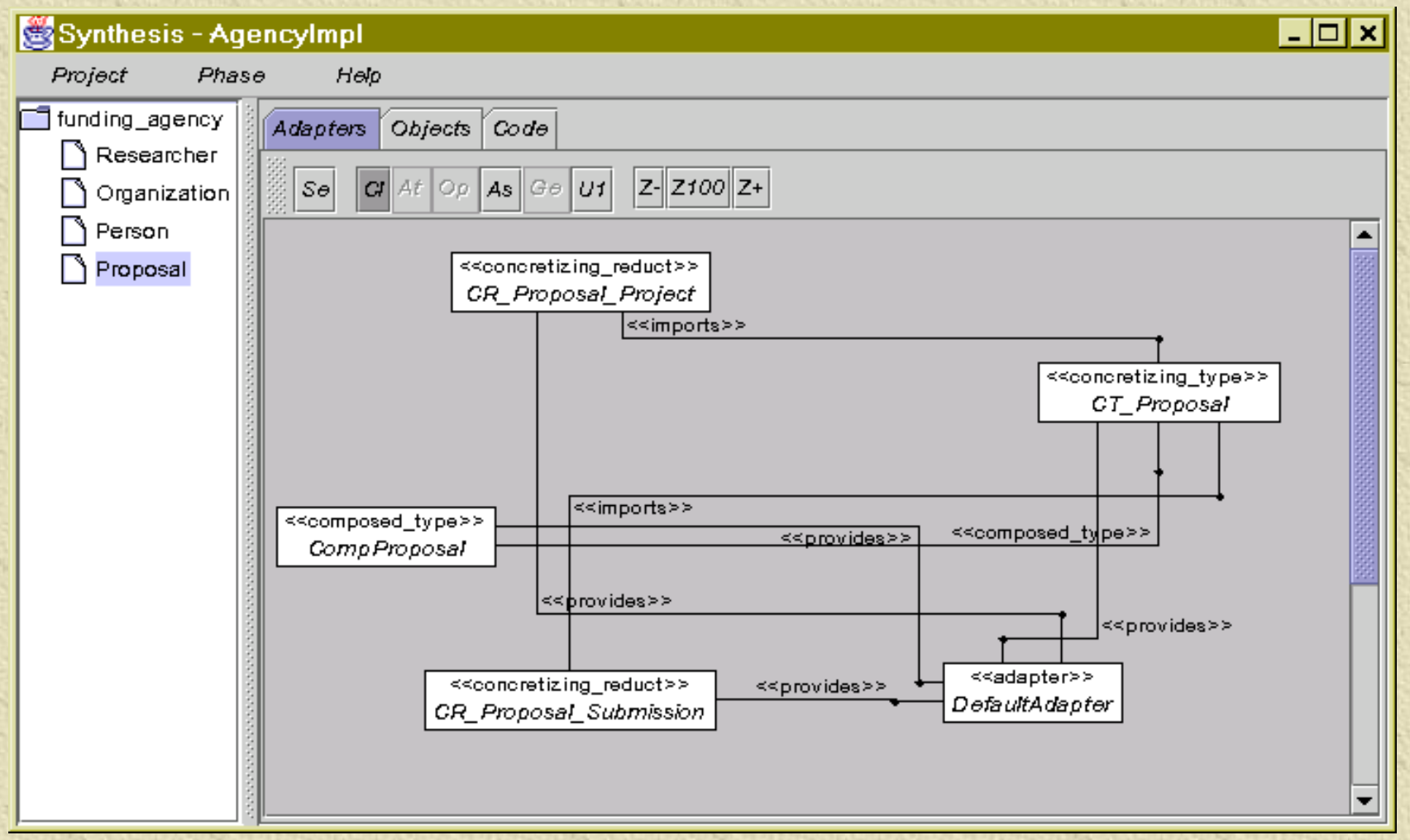
accept concretizing types

Этап реализации

- Генерация кода заключается в трансформации метаинформации, хранящейся в репозитории, в программы на каком-либо языке программирования.
- Каждому *типу этапа конструирования* соответствует один тип реализации.
- *Адаптер* обозначает группу типов реализации и служит единицей развертывания (deployment).
- *Объекты этапа реализации* - это конкрентные объекты, о существовании которых известно заранее.
  - Такие объекты обычно служат точками входа в систему и регистрируются, например в Naming Service.

Этап реализации

- Перед тем как начать генерацию кода, пользователь должен выбрать целевой язык программирования, а также целевые технологии, например, пользователь может выбрать технологию технической поддержки интероперабельности - RMI, CORBA или DCOM.
- В процессе генерации создаются следующие спецификации: IDL-интерфейсы (типы, factories), Java-классы (реализации типов, адаптеров, фабрик), make-файлы, а также происходит регистрация объектов в Naming Service и связывание с ними.


Synthesis - AgencyImpl
Project
Phase
Help
funding_agency
Researcher
Organization
Person
Proposal
Adapters
Objects
Code
Se
Cl
At
Op
As
Ge
U1
Z-
Z100
Z+
<<concretizing_reduct>>
CR_Proposal_Project
<<imports>>
<<concretizing_type>>
CT_Proposal
<<imports>>
<<composed_type>>
CompProposal
<<provides>>
<<composed_type>>
<<provides>>
<<provides>>
<<concretizing_reduct>>
CR_Proposal_Submission
<<provides>>
<<adapter>>
DefaultAdapter

Synthesis - AgencyImpl

Project Phase Help

- funding_agency
  - Researcher
  - Organization
  - Person
  - Proposal

Adapters Objects Code

Save CR_Proposal_Submission.get_staff

```
public Researcher[] get_staff(industrial_labs.Submission object)
{
industrial_labs.Employee[] team=object.team();
java.util.List lSpecialist=new java.util.ArrayList();
for(int i=team.length;i-->0;){
                industrial_labs.Employee employee=team[i];
                industrial_labs.Specialist specialist=
                                industrial_labs.SpecialistHelper.narrow(employ
                if(specialist!=null)
                                lSpecialist.add(specialist);
}//for
industrial_labs.Specialist[]aSpecialist=new industrial_labs.Specialist[lSpecia
lSpecialist.toArray(aSpecialist);
return Adapter.getCR_Researcher_SpecialistFactory().toAppSet(aSpecialis
}
```

Области применения

- электронные библиотеки
  - регистрации различных неоднородных коллекций в предметном посреднике
  - проектирования персонализированных виртуальных электронных библиотек
- Web Services
  - использование Web Services в качестве компонентов для их повторного использования в разрабатываемых ИС
- GRID