Соотношение SObjectizer и FIPA Abstract Architecture

Евгений Охотников
Intervale

31 мая 2006 г.

Содержание

1 Введение

SObjectizer [SO4] является фреймворком для агентно-ориентированного программирования на C++. Однако, термин *агентно-ориентированный* в настоящее время несколько перегружен. В частности, существует организация Foundation for Intelligent Physical Agents (FIPA) [FIPA], которая вырабатывает спецификации для мультиагентных систем (Multi Agent Systems – MAS). Во главу угла в MAS также ставится понятие агента. Но под агентами в MAS понимаются иные сущности, нежели в SObjectizer. Данная статья является попыткой провести небольшое сравнение агентов

SObjectizer и агентов FIPA на основе информации, описанной в спецификации FIPA Abstract Architecture [FIPA00001].

Примечание. Статья требует от читателя знания SObjectizer [SOBOOK] (хотя бы общих принципов его работы). Так же желательно иметь общее представление о FIPA, т.к. сама статья делает только очень поверхностный обзор FIPA Abstract Architecture.

2 FIPA Abstract Architecture

2.1 Цели и область действия FIPA Abstract Architecture

В спецификации FIPA Abstract Architecture об истории и целях FIPA Abstract Architecture говорится следующее:

> Целью FIPA является создание стандартов для агентов для продвижения интероперабельных агентных приложений и агентных систем. В 1997 и 1998 годах FIPA выпустила серию спецификаций агентных систем в которых интероперабельность агентых систем была основой. Эта работа включала в себя спецификации для агентной инфраструктуры и агентых приложений. Спецификация агентной архитектуры включала в себя язык коммуникации агентов (agent communiction language – ACL), сервисы агентов (agent services), и поддерживающие их управляющие онтологии (ontologies).
>
> В сердце модели агентных систем FIPA находится коммуникация агентов, где агенты могут отсылать друг другу имеющие четкую семантику сообщения для достижения цели приложения.
>
> Агентные системы, построенные по со спецификациями FIPA 97 и 98, были способны к интероперабильности с агентными системами, построенными в соответствии с FIPA Abstract Architecture, через транспортные шлюзы, но с некоторыми ограничениями. Архитектура FIPA 2000 еще более приблизилась к FIPA Abstract Architecture и будет способна к полной интероперабельности через шлюзы. Общая цель этого архитектурного подхода в том, чтобы сделать возможным создание систем, которые прозрачно интегрируются в свои специфические вычислительные среды и имеют при этом возможность взаимодействия с агентными системами, находящимися в других средах.
>
> Основное внимание в FIPA Abstract Architecture уделяется созданию обмена сообщениями с четкой семантикой между агентами, которые могут использовать различные транспорты сообщений, различные языки коммуникации агентов или различные языки контента (content language). Это определяет набор точек потенциальной интероперабельности. В область действия FIPA архитектуры входит:
>
> * Модель сервисов и средств обнаружения сервисов, доступных для агентов и других сервисов.
>
> * Интероперабельный транспорт сообщений.
>
> * Поддержка различных способов представления ACL.
>
> * Поддержка различных форм языков контента.
>
> * Поддержка различных форм представления каталогов (директорий) сервисов.

Иными словами, целью FIPA Abstract Architecture является описание некой абстрактной архитектуры, в которой возможно безболезненное и прозрачное взаимодейтсвие агентных систем, разработанных на разных языках программирования, работающих на разных платформах, использующих различные транспортные протоколы и т.д. Эта абстрактная архитектура не может быть реализована один-в-один. Кроме того, она не освещает некоторые моменты (управление временем жизни агентов или мобильность агентов, к примеру), оставляя их на откуп производителям агентных платформ. Тем не менее, FIPA Abstract Architecture формирует основу для разработки конкретных спецификаций, на основе которых можно будет создавать конкретные программные решения.

Основное внимание в FIPA Abstract Architecture уделяется интероперабильности агентов. Поэтому центральное место в FIPA Abstract Architecture занимают такие вещи, как:

- Управление различными схемами передачи сообщений.
- Управление различными схемами представления сообщений.
- Поиск агентов и сервисов в каталогах (директориях) сервисов.

2.2 Обзор архитектуры FIPA

FIPA Abstract Architecture определяет (на высоком уровне абстракции), как агенты могут находить друг друга и взаимодействовать между собой посредством регистрации и обмена сообщениями. Далее коротко описываются некоторые ключевые моменты архитектуры FIPA.

2.2.1 Агенты и сервисы

Агенты (**agents**) взаимодействуют путем обмена сообщениями (**messages**) которые выражены на языке коммуникации агентов (**agent-communication-language**).

Сервисы (**service**) предоставляют агентам услуги поддержки. FIPA определяет набор стандартных сервисов, в который входят такие сервисы, как сервис каталогов агентов (**agent-directory-service**) и сервис транспорта сообщений (**message-transport-service**).

Agent-directory-service необходим для того, чтобы после старта агенты могли зарегистрировать самих себя и найти интересующих их агентов для организации взаимодействия. Для регистрации агент оформляет элемент каталога агентов (**agent-directory-entry**), который передается в **agent-directory-service**. Как результат поиска агента в **agent-directory-service** возвращается **agent-directory-entry** с описанием найденого агента. **Message-transport-service** необходим для осуществления передачи сообщений через конкретные виды транспортов (**transport**), т.е. через разные коммуникационные протоколы.

При старте агент должен сначала выбрать для себя один или несколько **transport**-ов. Например, он может подключиться к CORBA ORB или начать прослушивать какую-нибудь очередь сообщений. После этого агент становится доступным через выбранные им **transport**-ы (говорят, что агент связался с **message-transport-service**). При этом агент получает набор описателей транспорта (**transport-descriptions**), которые содержат информацию о способе доступа к этому агенту через данный транспорт (например, если транспортом является HTTP, то в качестве **transport-description** будет выступать URL

для агента; если транспортом является SMTP, то **transport-description** будет адрес e-mail агента). Затем агент должен зарегистрироваться в **agent-directory-service**, передав в каталог свое имя (**agent-name**) и адрес (**agent-locator**, в котором перечисляются все **transport-descriptor** этого агента), а также набор необязательных атрибутов, описывающих агента.

Когда агент хочет найти другого агента для взаимодействия, он обращается в **agent-directory-service**. В результате поиска агент получает **agent-name** и **agent-locator** нужного ему агента. После чего становится возможным обмен сообщениями между ними.

2.2.2 Сообщения и транспорт

Сообщение (**message**) пишется на языке взаимодействия агентов (**agent-communication-language**) и состоит из содержимого (**content**), имени отправителя (**sender**), имен получателей (**receivers**) и, возможно, дополнительного набора атрибутов. Содержимое сообщения выражается на конкретном языке контента (**content-language**).

При передаче сообщение кодируется в пакет (**payload**), который включается в состав транспортного сообщения (**transport-message**). **Transport-message** затем пересылается посредством **message-transport-service** через конкретный **transport**. **Payload** формируется службой кодирования (**encoding-service**) в соотвествии с описанием представления (**encoding-representation**) для конкретного типа транспорта. Сам же **payload** внутри транспортного сообщения содержится внутри конверта (**envelope**) с необходимой для транспорта информацией.

2.2.3 Упрощенный пример

Образно говоря, если два агента хотят обмениваться между собой информацией о биржевых котировках, они:

- выбирают язык контента (один из специфицированных FIPA или собственный);
- определяют тип транспорта (например, XML поверх HTTP);
- создают у себя конкретные экземпляры **transport** и **encoding-service**, указывая конкретный **encoding-representation** для конкретного экземпляра **transport**;
- регистрируют себя в некотором **agent-directory-service** и находят друг друга с его помощью;
- один из агентов формирует **content** сообщения на выбранном ими **content-language** после чего создает экземпляр **message** на конкретном **agent-communication-language**, включая в него **content**, **sender** и **receiver**;
- экземпляр **message** преобразуется в **payload** посредством **encoding-service** для конкретного транспорта в соотвествии с **encoding-representation** этого транспорта;
- **payload** поступает к **transport**, из него формируется **transport-message** и готовый **transport-message** пересылается удаленному агенту;
- на принимающей стороне происходят обратные преобразования из **transport-message** в **payload**, из **payload** в **message**. В результате агент-получатель принимает **message** с нужным ему **content** (выраженном на нужном **content-language**).

2.3 Определение агента в FIPA Abstract Architecture

Для проведения сравнения SObjectizer и FIPA необходимо привести точное определение понятия агента из спецификации FIPA Abstract Architecture:

> **Резюме**
>
> Агент — это вычислительный процесс, который реализует автономную коммуникативную функциональность приложения. Обычно, агенты взаимодействуют посредством языка коммуникации агентов. Конкретное инстанцирование агента является обязательной частью каждой конкретной реализации FIPA Abstract Architecture.
>
> **Взаимосвязь с другими элементами**
>
> Агент имеет имя (**agent-name**).
>
> Агент может иметь атрибуты (**agent-attributes**).
>
> Агент имеет адрес (**agent-locator**), который перечисляет описания транспортов (**transport-descriptions**) для этого агента.
>
> Агент может отсылать сообщения (**message**) на **transport-description** через **transport**, соответствующий этому **transport-description**.
>
> Агент может отсылать **transport-message** одному или нескольким агентам.
>
> Агент может зарегистрироваться в одном или нескольких **agent-directory-services**.
>
> Агент может иметь **agent-directory-entry**, который регистрируется в **agent-directory-services**.
>
> Агент может модифицировать **agent-directory-entry**, зарегистрированный ранее в **agent-directory-service**.
>
> Агент может дерегистрировать свой **agent-directory-entry** из **agent-directory-service**.
>
> Агент может искать **agent-directory-entries** зарегистрированные в **agent-directory-service**.
>
> Агент может адресоваться через механизмы, описанные в **transport-descriptions** в его **agent-directory-entry**.
>
> **Описание**
>
> В конкретном воплощении FIPA Abstract Architecture агент может быть реализован разными способами, например, как компонент на Java, как COM объект, самостоятельной Lisp программой или TCL скриптом. Он может исполняться как нативный процесс на некотором физическом компьютере под управлением операционной системы, или поддерживаться интерпретатором, таким как Java Virtual Machine или TCL системой. Взаимосвязь между агентами и их вычислительными контекстами специфицируется в процессе жизненого цикла агента. FIPA Abstract Architecture не затрагивает жизненый цикл агента, т.к. часто он реализуется по разному в конкретных вычислительных средах. Реализация FIPA Abstract Architecture должна описывать эти моменты.

В этом определении самым важным является то, что агентом является самостоятельный вычислительный процесс, который существует внутри некоторой вычислительной среды. При этом непосредственной обязанностью данного процесса является поддержание коммуникации с другими агентами.

3 SObjectizer vs FIPA

Уже из приведенного выше очень краткого описания FIPA Abstract Architecture становится очевидной разница в масштабах целей SObjectizer и FIPA Abstract Architecture. SObjectizer предназначен для реализации **одного приложения** в виде взаимодействующих агентов. FIPA Abstract Architecture предназначена для реализации мультиагентных систем, в которых каждый из агентов является **самостоятельным приложением**.

Возможности SObjectizer по построению распределенных приложений (через механизм глобальных агентов и SOP транспорты) не приближают SObjectizer к FIPA Abstract Architecture, поскольку в таких приложениях для прикладных агентов создается иллюзия, что они являются частью одного монолитного приложения, работающего на одном компьютере.

Следовательно, можно сделать вывод о бесполезности сравнения SObjectizer и FIPA Abstract Architecture, поскольку они явно находятся в разных весовых категориях и предназначены для решения совершенно разных задач. Тем не менее, и в SObjectizer и в FIPA Abstract Architecture прослеживаются общие понятия и приемы. Поэтому интересно заострить внимание на некоторых вещах и показать, как отдельные понятия и механизмы FIPA Abstract Architecture видоизменяются в SObjectizer.

3.1 Соотношение на уровне философии

3.1.1 Платформы и языки

Целью FIPA является выработка спецификаций, на основе которых можно будет строить интероперабельные системы агентов. Поэтому ориентация на разные платфомы, языки и типы транспорта является идеологической основой FIPA Abstract Architecture. SObjectizer же изначально предназначался для упрощения написания отдельного класса приложений на C++. Поскольку C++ является кроссплатформенным языком, существующим на большом количестве платформ, то SObjectizer также разрабатывался как переносимый фреймворк чтобы предоставить пользователям SObjectizer возможность разрабатывать свои приложения на разных платформах.

Но при разработке SObjectizer поддержка нескольких языков программирования не была основной целью, поскольку SObjectizer, в отличии от FIPA Abstract Architecture, является конкретным инструментом для решения конкретных задач. От инструмента требуется наличие и доступность здесь и сейчас. Поскольку разработчики SObjectizer использовали в качестве основного языка C++, то и SObjectizer прежде всего создавался для C++. Поэтому в настоящее время SObjectizer, в отличии от FIPA Abstract Architecture, является моноязыковым инструментом.

Вопрос о поддержке мультиязычности в SObjectizer является вопросом о том, будет ли портирован SObjectizer на другие языки. Технически это возможно. В минимальном варианте необходимо портировать на новый язык систему сериализации ObjESSty [OESS] и реализовать поддержку SOP. Но более желательно иметь в новом языке такие

же понятия, как SObjectizer Run-Time, кооперация агентов, агент-коммуникатор и транспортный агент. Все эти технические вопросы не очень сложны и не слишком трудоемки в реализации. Главный вопрос политический: нужен ли SObjectizer на каком-нибудь языке, кроме C++? Ответ на него могут дать только пользователи SObjectizer проявляя заинтересованность в портировании SObjectizer для какого-нибудь языка/платформы.

Резюме можно выразить следующим образом: FIPA говорит, что поддержка мультиязычности в FIPA Abstract Architecture — критически необходима. SObjectizer же говорит, что поддержка мультиязычности вполне возможна, если только эта поддержка будет кому-нибудь необходима.

3.1.2 Агенты

Несмотря на то, что SObjectizer и FIPA Abstract Architecture находятся в разных весовых категориях, в обеих системах эксплуатируется одна и та же идея: приложение формируется из отдельных автономных агентов, отвечающих за решение конкретных проблем и выполнение конкретных задач. Только в SObjectizer приложением может быть один процесс, в то время как в FIPA Abstract Architecture под приложением будет пониматься совокупность более мелких автономных приложений. Однако принцип остается одинаковым: большую функциональность можно получить путем объединения агентов с меньшей функциональностью.

Например, в разработанных с помощью SObjectizer системах применяется несколько повторно используемых компонентов, таких как подсистема динамического конфигурирования приложения, подсистема логирования, подсистема мониторинга, подсистема согласованной доставки сообщений и др. Каждая из подсистем состоит из одного или нескольких агентов, за регистрацию которых отвечает пользователь. Однако, после своей регистрации каждый агент начинает жить собственной жизнью, по собственным правилам. Тем самым наполняя приложение нужной функциональностью. Так, подсистема логирования обрабатывает сообщения о ходе работы приложения. Подсистема мониторинга обрабатывает информацию об изменении отдельных величин в приложении (например, таких как показания датчиков температуры в системе климат-контроля). Подсистемы мониторинга и логирования работают параллельно друг другу, но ничего не знают друг о друге. Более того, прикладная часть приложения может не знать об их присутствии, но тем не менее, будет пользоваться их функциональностью.

Побочным эффектом такой архитектуры является то, что построенное на основе агентов приложение напоминает движение муравьев в муравейнике или летящих стаей птиц. На первый взгляд взаимодействие всех агентов в приложении и объемы пересылаемых сообщений будут напоминать хаотическое движение (как при поверхностном взгляде на муравьев или птиц). И временами может казаться, что приложение абсолютно непредсказуемое, живущее по каким-то своим собственным законам. Отчасти так и есть, однако законы эти вполне определены — это логика, заложенная в агентов при реализации.

Что же касается логики, то SObjectizer подразумевает, что агенты будут представляться конечными автоматами и предлагает для этого соответствующую поддержку в виде состояний агентов и привязки событий к состояниям. Кроме того, в SObjectizer подразумевается, что агенты не будут выполнять сложных операций по поиску собеседников для обмена сообщениями. В подавляющем большинстве случаев все коммуникации заранее определены разработчиком SObjectizer приложения, и агенты

просто изначально знают имена тех, с кем им предстоит взаимодействовать. Поэтому можно сказать, что агенты в SObjectizer менее интеллектуальны, чем в FIPA.

3.1.3 Потоки данных

Идеология FIPA подразумевает, что между агентами нет заранее жестко определенных связей. Считается, что связи есть между типами агентов на уровне логики (можно сказать на уровне типов **content-language**). Например, при разработке агента для отслеживания биржевых рисков становится понятно, что ему потребуется функциональность агентов, предоставляющих биржевые котировки, и агентов, выполняющих некий статистический анализ. Но ни на этапе разработки агента, ни даже на этапе его развертывания в принципе не известны имена и адреса агентов, с которыми он будет взаимодействовать в реальной жизни. Предполагается, что конкретные информационные связи агент установит сам после своего старта путем поиска подходящих кандидатов в соответствующих **agent-directory-services**.

При этом может оказаться, что информационные связи будут динамическими и смогут изменяться не только от запуска к запуску, но и в процессе непрерывной работы агента. Например, пусть изначально агент установил связь с агентами www.superstock.com/monitor и www.superstat.com/calc. С течением времени агент может обнаружить более дешевого поставщика биржевых котировок www.cheapstock.biz/verycheap и переключиться на работу с ним. А агент www.superstat.com/calc может перестать функционировать и вместо него придется переключиться на www.allstatmethods.com/expensive. Т.е. для FIPA правилом является то, что агенты сами устанавливают направления потоков данных и сами отвечают за их переконфигурацию.

В случае с SObjectizer ситуация обратная. Основная задача SObjectizer состоит в облегчении создания некоторых типов приложений. А при проектировании приложения разработчик заранее знает из каких частей приложение состоит, какие взаимоотношения есть между отдельными частями. Следовательно, потоки данных в приложении определены заранее и не нуждаются в переконфигурировании по ходу работы.

Поэтому можно сказать, что в FIPA агенты отвечают за создание потоков данных и за последующее управление этими потоками. В то время как в SObjectizer агенты отвечают только за обработку уже существующих потоков данных.

3.2 Соотношение на уровне архитектуры

3.2.1 Агенты и сервисы

Как и в FIPA, в SObjectizer агенты взаимодействуют посредством обмена сообщениями. Но в SObjectizer сообщениями являются обычные C++ классы. Нет таких понятий, как **content-language** и **agent-communication-language**. Нет такого ярковыраженного понятия, как онтология (**ontology**) — в SObjectizer каждое сообщение имеет собственную смысловую нагрузку и сообщения одного агента также образуют некоторую онтологию. Но в SObjectizer для онтологии нельзя задать имя (именем может считаться имя типа, к которому принадлежит агент-владелец сообщений). Также нельзя в динамике определить, какую именно онтологию поддерживает конкретный агент.

В SObjectizer нет понятия сервисов (**services**). Вместо этого есть SObjectizer Run-Time, который выполняет часть работы, отводимой в FIPA сервисам. Для некоторых

целей (например, транспорт сообщений, как будет показано ниже) возможности Run-Time расширяются посредством специальных диспетчеров и вспомогательных агентов (транспортных), за создание которых отвечает пользователь SObjectizer.

3.2.2 Отсутствие agent-directory-service

В SObjectizer нет ничего, что бы можно было сравнить с **agent-directory-service**. Run-Time отвечает за регистрацию/дерегистрацию агентов и за хранение системного словаря (каталога в смысле FIPA Abstract Architecture). Но в SObjectizer, в отличие от FIPA Abstract Architecture, агент не может заниматься поиском агентов в каталоге — агент должен точно знать своих собеседников заранее.

3.2.3 Транспорт сообщений

В SObjectizer нет такой стройной и гибкой архитектуры для транспорта сообщений, как в FIPA Abstract Architecture. Нет понятий **transport**, **encoding-service**, **encoding-representation**, **message-transport-service**. Вместо этого есть транспортные агенты (аналоги **transport**), задачей которых является поддержание коммуникационного канала через конкретный механизм Inter-Process Communication (IPC). Например, такой как TCP-сокеты, разделяемая память, Unix pipes. Вместо **encoding-service** и **encoding-representation** есть двоичная сериализация C++ объектов средствами ObjESSty. SObjectizer использует один способ сериализации для любого транспорта, поэтому надобности в **encoding-representation** нет. В качестве **encoding-service** можно представить агента-коммуникатора, который в SObjectizer отвечает за сериализацию/десериализацию сообщений, идущих через SOP-каналы. В качестве **message-transport-service** можно рассматривать связку агента-коммуникатора с транспортными агентами, т.к. именно эта связка обеспечивает транспорт сообщений в распределенных SObjectizer приложениях.

3.2.4 Отсутствие transport-description

В SObjectizer нет понятия **transport-description**. Во-первых, пересекать границы процесса в SObjectizer могут только сообщения глобальных агентов. А глобальный объект — это такое понятие, которое обеспечивает присутствие глобального агента одновременно во всех частях распределенного приложения. Во-вторых, сообщения глобальных агентов распространяются только через SOP каналы, при этом не важно, поверх какого IPC транспорта построен канал. В результате, распределенное SObjectizer приложение может рассматриваться как одно большое приложение, в котором сообщения передвигаются совершенно прозрачно, не обращая внимания на границы процессов и виды транспорта.

Для адресации агентов в SObjectizer используются имена агентов, не зависящие от типа транспорта, через который агент доступен. Для отсылки сообщения агента нужно всего лишь отправить на его имя сообщение. SObjectizer переправит сообщение во все доступные каналы и, если на принимающей стороне есть агент с указанным именем, то сообщение будет ему доставлено. У каждого коммуникационного SOP канала есть свой идентификатор канала. Этот идентификатор можно использовать для организации peer-to-peer взаимодействия агентов: если при отсылке сообщения указан идентификатор канала, то SObjectizer доставит сообщение только через этот SOP-канал, а остальные каналы будут проигнорированы. В этом смысле пара из имени агента и идентификатора

канала может быть аналогом **transport-description**. Но если в FIPA Abstract Architecture **transport-description** требуется для любого взаимодействия между агентами, то в SObjectizer идентификатор канала используется только для peer-to-peer коммуникаций.

3.2.5 Детали формирования transport-message

Спецификация FIPA Abstract Architecture уделяет большое внимание принципам формирования **transport-message**: специально выделены отдельные понятия **payload**, **content**, **sender**, **receiver**, **transport-message** и пр. Возможно, в конкретных реализациях FIPA пользователи не сталкиваются напрямую с подобными вещами, и все детали формирования **transport-message** берет на себя инструментальная система. Тем не менее, в FIPA Abstract Architecture перечисленным выше понятиям отводится очень важная роль в обеспечении интероперабельности различных воплощений FIPA Abstract Architecture.

В SObjectizer аналоги подобных понятий также существуют на уровне SOP. Но в SObjectizer они являются деталями реализации, о которых пользователю не сообщается вообще. Целью транспортных агентов и SOP протокола является предоставление пользователю полной прозрачности движения сообщений между SObjectizer процессами. Пользователь должен просто отсылать и получать сообщения в виде экземпляров C++ классов, все остальное (как то: выбор транспорта, выбор конкретного **encoding-representation**, использование конкретного **encoding-service**, создание **transport-message**) от пользователя скрыто в SObjectizer Run-Time.

Нужно оговориться, что подобное сокрытие деталей формирования **transport-message** в SObjectizer тривиально, поскольку SObjectizer на данный момент является моноязыковым инструментом и использует всего один **encoding-service** (сериализация средствами ObjESSty). Возможно, степень прозрачности обмена сообщениями между процессами в SObjectizer существенно снизится когда SObjectizer будет адаптирован для нескольких языков программирования.

3.2.6 Упрощенный пример

Пример, приведенный в 2.2.3, для SObjectizer будет выглядеть следующим образом (подразумевается, что агенты находятся в разных процессах распределенного приложения):

- описывается глобальный агент, сообщения которого будут использоваться для взаимодействия прикладных агентов (это можно считать аналогом выбора онтологии, языка контента и языка коммуникации агентов в FIPA);
- в каждом приложении глобальный агент специальным образом регистрируется в SObjectizer Run-Time. Таким образом получается, что глобальный агент как бы существует одновременно во всем распределенном приложении, без учета границ процессов;
- в каждом приложении прикладные агенты подписываются на сообщения глобальных агентов;
- в каждом приложении создаются транспортные агенты для организации IPC каналов. Транспортные агенты отвечают за установление соединения, его контроль и восстановление в случае сбоев;

- для взаимодействия друг с другом прикладные агенты просто отсылают сообщения глобального агента. И получают их через обычный механизм обработки событий в SObjectizer.

Интересной особенностью SObjectizer приложений является то, что создание транспортных агентов не является частью бизнес логики приложений. Т.е. вполне возможно реализовать прикладных агентов так, чтобы они считали, что транспорт им обеспечит кто-то другой (например, подсистема динамического конфигурирования приложения способна по описаниям в конфигурационных файлах создавать нужное количество транспортных агентов с нужными параметрами). В этом случае шаг по созданию транспорта можно считать деталью конфигурирования, а бизнес логика будет состоять всего лишь из: описания глобального агента, регистрации глобального агента, подписки на сообщения глобального агента и обмена сообщениями глобального агента. Но при таком взгляде на приложение не важно, находятся ли прикладные агенты в одном процессе или нет. Что позволяет конфигурировать приложение исходя из конкретных обстоятельств. Например прикладных агентов можно разместить на разных узлах сети в разных процессах для обеспечения надежности, А можно и в одном процессе, если коммуникации становятся узким местом. Важно, что для самих прикладных агентов подобное перемещение совершенно прозрачно.

4 Заключение

FIPA и SObjectizer являются технологиями из разных весовых категорий. FIPA предназначена для создания MAS, в которых каждый агент представляется отдельным приложением, написанным на каком-то языке программирования и работающим на своей вычислительной платформе. Целью FIPA является обеспечение интероперабильности между различными реализациями FIPA Abstract Architecture. Поэтому в FIPA ставится акцент на наличие различных видов транспорта сообщений и на обеспечение интероперабильности взаимодействия агентов для чего введены такие понятия, как **transport**, **transport-description**, **encoding-representation**, **payload**, **content**, **transport-message**, **agent-name**, **agent-location**, **agent-directory-service** и др.

SObjectizer, в первую очередь, является инструментом для разработки многопоточных приложений на C++, в которых отдельные сущности выгодно представлять в виде конечных автоматов. Возможность построения распределенных приложений средствами SObjectizer явилась побочным эффектом механизма взаимодействия агентов исключительно через обмен сообщениями. Эта возможность была развита таким образом, чтобы обеспечить агентам прозрачность взаимодействия без учета границ процессов и распределения агентов по процессам. Также в настоящее время SObjectizer является моноязыковым инструментом, поэтому для него нет вопросов по обеспечению интероперабильности между различными реализациями SObjectizer на различных языках программирования.

FIPA и SObjectizer используют одну и ту же идею о построении приложения (в FIPA можно говорить о системе приложений) из отдельных автономных сущностей – агентов, обладающих собственной логикой и собственными правилами поведения. По сравнению с FIPA агенты в SObjectizer менее интелектуальны, т.к. освобождены от части задач, которые вынуждены решать агенты FIPA.

Тем не менее в FIPA Abstract Architecture и SObjectizer можно обнаружить много совпадений и похожих решений. Поэтому можно было бы сравнить SObjectizer с микро-

вариантом FIPA Abstract Architecture, ориентированным на создание одного приложения (возможно распределенного). И, за счет этого, существенно более упрощенного варианта. Однако, ключевым отличием останется то, что в FIPA Abstract Architecture агенты вынуждены знать о том, что между ними есть транспорты и, что эти транспорты могут быть разными. В SObjectizer упор делается на обеспечение прозрачности взаимодействия, поэтому, в идеале, агенты вообще не должны знать о наличии какого-либо транспорта между ними.

Список литературы

[SO4] `http://sobjectizer.sourceforge.net`

[FIPA] `http://www.fipa.org`

[FIPA00001] FIPA Abstract Architecture Specification. Foundation for Intelligent Physical Agents, 2002. `http://www.fipa.org/specs/fipa00001/SC00001L.html`

[SOBOOK] Е. Охотников. SObjectizer-4 Book, 2004. `http://eao197.narod.ru/desc/so_4_book.pdf`

[OESS] `http://eao197.narod.ru/objessty`