Chapter 1
Agent-Based Computation of Decomposition Games with Application in Software Requirements Decomposition

Jiamou Liu, Ziheng Wei

Abstract Coalition formation is a fundamental question in multiagent systems. The question asks for an optimal way in which agents may form coalitions and cooperate to accomplish a task. In this paper we investigate the use of coalition formation in software architecture design. We investigate a multiagent framework for attribute-driven software architecture design process. We design an agent for each requirement; the agents form coalitions that represent software components. The coalition formation process is based on decomposition game, a variant of coalition game. We extend previous work by adopting the propose-select-adjust framework for computing solutions of decomposition games. The focus is on analysing efficiency and utility of this agent-based approach. We also present three real-world case studies demonstrating the use of this approach to support software architecture design.

Key words: Coalition formation, decomposition game, software architecture, software requirements, PSA-framework

1.1 Introduction

Coalition formation is the problem of *cooperation*: In a multiagent environment, the outcome of a task can often be improved if several agents join force to accomplish it together. The problem is important in a wide range of application domains such as task allocation [10] and electronic markets [13]. Therefore coalition formation has been a major topic of interest in multiagent systems [11]. *Coalition games* are useful tools to investigate coalition formation. Such games consist of a number of players, any non-empty subset of which is called a possible *coalition*. For each possible coalition, the game assigns a real number expressing its utility, which would then

Jiamou Liu[1] · Ziheng Wei[2]
Department of Computer Science, The University of Auckland, Auckland, New Zealand
[1]e-mail: `jiamou.liu@auckland.ac.nz` [2]e-mail: `zwei891@aucklanduni.ac.nz`

be used to derive payoffs of members of the coalition. The output of the game is a partition of all players, called a *coalition structure*, which should satisfy certain *stability* criterion, such as core, kernel, and Shapley value; these stability concepts all mean — to a degree — that the output coalition structure represents an equilibrium where the payoff of all agents are well-balanced and fair [1].

In this paper we investigate coalition formation as applied to the problem of *software architecture design*. Software architecture forms an important bridge between requirement analysis and more concrete software designs in the software engineering process; it defines a high-level software composition which meets function and quality requirements. The design and evaluation of software architectures typically demands high-level expertise and amounts to a largely manual task [3].

In our previous work [9], we initiated a game-based study of software architectures, hence providing a formal basis that supports the design and evaluation of software architectures. Our intuition is this: Software requirements exhibit *conflicts* and it is not possible to fulfill all requirements to a perfect degree; take, for example, security and performance, both are important quality feature of a software system. However, to ensure security, layers of encryption mechanisms should be put in place of a software system which harms performance. Therefore instead of finding the "perfect" software architecture, one would usually aim to find a "reasonable" software architecture that nicely balance all requirements. Abstractly, we may view software architecture design as a game, where players are requirements that may or may not be at odds of each other, and whose solutions are certain equilibrium states which satisfy all requirement to the best degree; such solution corresponds to a reasonable software architecture. Our approach blends two novel ideas:

(a) Firstly, we proposed a new game model, called a *decomposition game*, which follows the general setup of coalition games, but with the following important exceptions: Players are altruistic in the sense that they aim to maximise the utility of their coalitions, rather than their individual payoffs; the utility of a coalition depends solely on the interactions between its members which may either be beneficial or detrimental. The associated solution concept is called *rational decomposition*, and is computed using centralised algorithms.
(b) Secondly, we modeled the process of *attribute-driven design* — a method transforming software requirements to a conceptual software architecture — using decomposition games. Players of the game are software requirements and utility of a software component of is captured by the set of requirements it meets. The process of software architecture design is thus a process of coalition formation: Starting from the coalition of all requirements, we recursively divide the current collection into smaller coalitions, which are eventually mapped to meaningful software components.

The work [9] leaves some important questions unanswered. The first question concerns with the approach of finding a rational decomposition. A game entails that players are "acting on their own wills"; hence it is natural to ask if a multiagent approach (instead of a centralised approach) can be used to simulate the formation of coalitions. The second question concerns with the complexity of the computation.

The third questions concerns with how well the proposed approach may be adopted in practice. In this paper we aim to answer these questions:

1. We develop a multiagent environment for computing rational decompositions. Here we utilise the propose-select-adjust *framework* introduced in [7] and subsequently developed in [8, 2]. The approach involves two types of agents, coalitions and sub-coalitions, who autonomously decide on a coalition structure.
2. To achieve efficient computation, we pose a constraint T, which denotes the total amount of queries an agent is allowed to make within an iteration. This constraint helps to reduce the workload of a single agent in any iteration to constant time, and hence improves computation time. We demonstrate through experiments that under such constraints one can still identify reasonable decompositions.
3. We discuss the design of three real-world software systems: SplitPay system, Wargame2000 system and Cafeteria ordering system. For each case study we describe how our approach may help to identify a rational software architecture. The case studies demonstrate our method as a viable approach for deriving and evaluating software architectures, as well as trade-off analysis.

The rest of the paper is organised as follows: Section 1.2 presents decomposition games as the theoretical basis of our mechanism for coalition formation. Section 1.3 introduces the PSA-framework for realising coalition formation in a distributed approach, where agents starts from singleton coalitions and perform merge and bind operations to form larger coalitions. Section 1.4 discusses application of this coalition formation process in software architecture design. Section 1.5 presents experiments using synthetic games. Section 1.6 concludes the paper with a discussion on future works.

1.2 Decomposition Games

Let N be a set of players. A *coalition* is a non-empty subset of N. A *coalition structure* on N is a collection of coalitions $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ such that $\bigcup_{i\geq 1}^{k} C_i = N$ and $C_i \cap C_j = \varnothing$ for any $i \neq j$. A *coalition game* models the situation when a group of agents form coalition based on individual payoffs; formally it is defined as a pair (N, v) where $v : 2^N \to \mathbb{R}$ is a *characteristic function* assigning every coalition to a utility. In the following, we consider a special type of coalition games, which have the following significant characteristics:

1. **Altruistic players**: We assume that players follow altruistic principle that values the collective utility of its coalition over individual payoffs. Therefore for any single player $a \in N$, the payoff of a equals to the utility of the coalition S that a belongs to. This is different from classical coalition games which assumes the sum of payoffs of members of a coalition S equals to the utility of S.

2. **Influence among players**: We assume that utility of a coalition is determined by the interactions among its members. For player a, there could be three types of influence to another player b: (1) *a benefits b*, i.e., a has a positive effect on b; (2) *a detriments b*, i.e., a has a negative effect on b; (3) a is *independent* from b, i.e., a has no effect on b. Furthermore, we assume asymmetric relation so that the influence from a to b may be of different from the influence from b to a.

We define a decomposition game as follows. Let E be the set of ordered pairs of distinct players in N. Let $E_p \subseteq E$ be a set of *positive influence edges*. Let $E_n \subseteq E$ be a set of *negative influence edges*. We require that $E_p \cap E_n = \varnothing$. The *influence matrix* σ assigns every pair $(a,b) \in \mathbb{N}^2$ a value in $\{-1,0,1\}$ such that $\sigma(a,b) = 1$ if $(a,b) \in E_p$, $\sigma(a,b) = -1$ if $(a,b) \in E_n$ and $\sigma(a,b) = 0$ otherwise. A *relevance function* is $w : N \times N \to \mathbb{R}$ which measures the relevance $w(a,b)$ between two players a,b; we require $w(a,b) = w(b,a)$. The *interaction function* $\rho : N^2 \times 2^N \to \mathbb{R}$ denotes the level of interactions from player a to a player b within a coalition S, taking into account the influence from a to b:

$$\rho(a,b,S) = \begin{cases} \sigma(a,b) \times \sum\limits_{a \neq c \in S} w(a,c), & \text{if } \sigma(a,b) \geq 0 \\ -\left| \sum\limits_{a \neq c \in S} w(a,c) \right|, & \text{otherwise} \end{cases}$$

Intuitively, the sum $\sum_{a \neq c \in S} w(a,c)$ denotes the relevance of a to the coalition S. If a is independent from b, then the interaction from a to b is 0; if a benefits b, then this sum is the level of interaction from a to b; if a detriments b, then the negative value of this sum if the level of interaction.

Definition 1 (Decomposition game). A *decomposition structure* is $D = (N, E_p, E_n, w)$ as described above. A *decomposition game* of D is a coalition game $G = (N, v)$ where $v(S) = \sum\limits_{a,b \in S} \rho(a,b,S)$

In a decomposition game (N,v), players form a coalition structure based on their payoffs. We view the formation of coalitions as a dynamic process that starts with all players in singleton coalitions. In other words, if $N = \{a_1, \ldots, a_n\}$, the starting coalition structure is $\mathscr{C}_0 = \{\{a_1\}, \{a_2\}, \ldots, \{a_n\}\}$ where $N = \{a_1, \ldots, a_n\}$. The players then choose to form bigger coalitions if they can obtain higher payoff in this way. Suppose at some point the players arrive at a coalition structure $\{C_1, C_2, \ldots, C_k\}$. We assume that any individual player a's knowledge is bounded within her own coalition. This means, $a \in N$ only have knowledge of subsets $S \subseteq C_i$ where $a \in C_i$. Thus the players and coalitions may perform two strategies:

- *Merge:* Several coalitions may choose to merge if they would obtain a higher combined utility than their respective utilities.
- *Bind:* Several players within the same coalition may form a sub-coalition if they would obtain a higher utility than the current coalition.

The outcome of the decomposition game (N,v) is a coalition structure in which neither merge nor bind may take place:

Definition 2 (Rational decomposition). Let $\mathscr{C}$ be a coalition structure of N in decomposition game (N, v).

1. $\mathscr{C}$ is *merge-free* if for all $\mathscr{S} \subseteq \mathscr{C}$, $v\left(\bigcup \mathscr{S}\right) \leq \max\{v(C) \mid C \in \mathscr{S}\}$
2. $\mathscr{C}$ is *bind-free* if for all $C \in \mathscr{C}$, any $S \subseteq C$, $v(S) \leq v(C)$.

Call the coalition structure $\mathscr{C}$ a *rational decomposition* if it is both merge-free and bind-free.

A rational decomposition always exists and may not be unique. From a computation point of view, however, computing a rational decomposition is NP-hard [9]. Thus it makes sense to find ways to approximate a rational decomposition.

1.3 A Multiagent Framework for Decomposition Games

The propose-select-adjust (PSA) framework is a decentralized framework for simulating decision making among a network of agents [7, 8]. Generally speaking, each agent in this framework can be considered a utility-based agent, which maintains and updates its own state according its perception about the performance of possible future states. To carry out its computation, each agent repeatedly follows a simple three-step action of propose, select, and adjust:

- propose: The agent derives a *proposal* based on environment
- select: The agent collects proposals from other agents and selects a proposal with the highest performance
- adjust: The agent updates its own state according to the selected proposal

The above process is repeated by all agents in the network simultaneously and indefinitely, and thus an agent in PSA can also be regarded as a cell in a *graph dynamical system* [8]. At any time instance, the global state of the network (i.e., the collections of states of all agents) reveals the collective solution of all agents. In [7, 2], PSA have been applied to the problems of community detection in social networks, as well as market segmentation in consumer-commodity networks. It has been revealed that PSA is a viable framework for simulating and monitoring evolutions in dynamic networks. In this paper, we use PSA as the computation framework for obtaining coalition structures in decomposition games. Our PSA framework contains two types of agents:

- **Coalition agents**: At any time instance, the decomposition game (N, v) is at a particular coalition structure $\mathscr{C} = (C_1, \ldots, C_k)$. Each coalition C_i is an agent which we call a *coalition agent*.
- **Sub-coalition agents**: For any coalition C_i, players in C_i also form a *sub-coalition structure* $\mathscr{D}_i = (D_1, \ldots, D_\ell)$ of C_i. Each sub-coalition D_i is also an agent which we call a *sub-coalition agent*.

The two types of agents perform different propose, select, adjust steps.

- For a coalition agent C_i, its proposal $P(C_i)$ consists of a set of coalition agents $\{C_{j_1},\ldots,C_{j_\ell}\}$ such that $v\left(C_i \cup \bigcup_{1\leq s\leq \ell} C_s\right) > v(C_i)$. A coalition agent would s-elect the proposal with the highest utility made by other coalition agents that include itself. Once it selects a proposal $P(C_j)$, the coalition merges itself with the coalition agent C_j to form a larger coalition.
- For a sub-coalition agent D_i, its proposal $P(D_i)$ consists of a set of sub-coalition agents $\{D_{j_1},\ldots,D_{j_\ell}\}$ such that each D_{j_s} and D_i belong to the same coalition, and $v\left(D_i \cup \bigcup_{1\leq s\leq \ell} D_s\right) > v(D_i)$. A sub-coalition agent would select the proposal with the highest utility made by other sub-coalition agents which include itself. Once it selects a proposal $P(D_j)$, the sub-coalition merges itself with the sub-coalition agent D_j to form a larger sub-coalition. As soon as any sub-coalition's utility exceeds the utility of the coalition that contains them, this sub-coalition "breaks away" from this coalition and lift itself into a coalition.

Figure 1.1 illustrates a possible scenario when two sub-coalitions bind to form a bigger sub-coalition and separate from their original coalition.

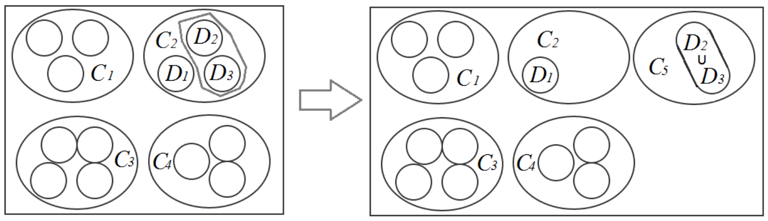

Fig. 1.1 Two sub-coalitions form a bigger sub-coalition whose utility exceeds their coalition, and thus breaks away to form a new coaltion.

Let $\mathscr{C} = \{C_1,\ldots,C_k\}$ be a coalition structure. An *optimal proposal* of a coalition C_i is a set $P_o(C_i) = \{C_{j_1},\ldots,C_{j_m}\}$ such that $v\left(C_i \cup \bigcup_{1\leq s\leq m} C_{j_s}\right)$ is maximal. For a sub-coalition structure $\mathscr{D}_i = \{D_1,\ldots,D_\ell\}$ of D_i, an *optimal proposal* $P_o(D_j)$ of a sub-coalition D_j can be defined in a similar way. The following theorem states that optimal proposals lead to the desired solution.

Theorem 1. *If the* PSA*-framework is implemented in such a way that all coalition and sub-coalition agents only make optimal proposals, then the players eventually stablise at a coalition structure $\mathscr{C}_o$. Furthermore, $\mathscr{C}_o$ is a rational decomposition.*

Searching for an optimal proposal for any agent may demand exhaustively looking through the space of all subsets of agents and thus is time-consuming. To ensure timely return of a proposal, we put a bound $T \in \mathbb{N}$ on the amount of information an agent is allowed to query in order to obtain a proposal. Under this constraint, a coalition agent C_i is not able to scan over arbitrary subsets of agents, but is restricted to examining only subsets of a bounded size α, defined as the largest value that satisfies $k^{\alpha} \leq T$, where $k = |\mathscr{C}|$, the number of coalitions in the current coalition structure $\mathscr{C}$. Thus we define the proposal $P_T(C_i)$ made by a coalition agent C_i as a subset of $\mathscr{C}$ with $|P_T(C_i)| \leq \alpha$ and

$$\forall P \subseteq \mathscr{C} : |P| \leq \alpha \Rightarrow v\left(C_i \cup \bigcup P_T(C_i)\right) \geq v\left(C_i \cup \bigcup P\right)$$

Similarly, a sub-coalition agent D within a sub-coalition structure $\mathscr{D}_i$ is not able to check arbitrary subsets of sub-coalitions in $\mathscr{D}_i$, but is restricted to examine subsets of a bounded size α, defined as the largest such that $\ell^{\alpha} \leq T$, where $\ell = |\mathscr{D}_i|$. We can then define the proposal $P_T(D)$ made by a sub-coalition agent D similarly to $P_T(C_i)$. The proposals $P_T(C_i)$ and $P_T(D)$ are referred to as *T-bounded optimal proposals*. The following is easy to check.

Proposition 1. *For a fixed $T \in \mathbb{N}$, computing a T-bounded optimal proposals for an agent takes constant time.*

Using T-bounded proposals to implement the PSA framework, the players would also reach a stablising coalition structure, which we call *T-bounded decompositions*. We will show using experiments in subsequent sections that T-bounded decompositions are very likely to be rational decompositions.

1.4 Game-Based Software Architecture Design

Decomposition game is introduced to formally capture the *attributed-driven design* (ADD) methodology. ADD amounts to a thorough and standardised paradigm for software architectures design after extensive development in the last 15 years [5][15]. Inputs to ADD are software requirements and their relations, which are assumed to be derived from requirement analysis. There are two types of requirements: *Functional requirements*, which specify tasks the system performs, and *non-functional requirements*, which refer to quality attributes such as performance, security, availability, modifiability, usability, testability, and portability. We describe these quality attributes using *general scenarios*. The actual non-functional requirements are instances of general scenarios, simply called *scenarios*, which are real-world situations that refer to specific general scenarios. For example, "*If a failure occurs, the banking system notifies the user; the system continues to perform with half the efficiency*" is a scenario that refers to the availability requirement.

Requirements exhibit complicated relations. For example, "the system finalises a payment transaction" should be preceded by "correct user credentials are checked" (this is a form of dependency between functional requirements), and quality attributes such as ensuring security of the system typically harm its usability. Requirement analysis has identified influences between common quality attributes; see [14] for full description of the influence matrix σ. We note that the influence matrix is not symmetric: e.g., An improvement in performance may not affect security, but increasing security will almost always adversely impact performance.

An *attribute primitive* is a software component that meets several functional and non-functional requirements. Examples of attribute primitives include data router, firewall, virtual machine, interpreter and so on. In particular the entire software can be regarded as an attribute primitive that meets all requirements. The ADD methodology specifies a list of common attribute primitive along with their properties and side effects; see for example in [4]. We use F to denote the set of functional requirements and S to denote the set of scenarios (non-functional requirements). Let $\mathsf{R} = \mathsf{F} \cup \mathsf{S}$. A *design element* is a subset $C \subseteq \mathsf{R}$. A *decomposition* of an attribute primitive is a set of design elements $\{C_1, C_2, \ldots, C_k\}$ which form a coalition structure of all requirements met by the attribute primitive.

The ADD process can be viewed as a process of requirement decomposition: The process starts with the whole software as an attribute primitive, and derives a decomposition $\{C_1, C_2, \ldots, C_k\}$ of R. The process then assigns an attribute primitive $\mathscr{A}_i$ that meets each design element C_i. If $\mathscr{A}_i$ in turn requires further decomposition, the ADD process will then be recursively applied to $\mathscr{A}_i$. Therefore, the ADD process is the problem of deriving a suitable decomposition given the set of requirements R and their relations. We describe the ADD procedure as applied to an attribute primitive $\mathscr{A}$ recursively as follows:

Procedure 1 $\mathsf{ADD}(\mathscr{A})$ (General Plan)

```
1: $(C_1, C_2, \ldots, C_k) \leftarrow \mathsf{Decompose}(\mathscr{A})$ // compute a rational decomposition of $\mathscr{A}$
2: for $1 \leq i \leq k$ do
3:     $\mathscr{A}_i \leftarrow$ an primitive attribute consistent with $D_i$
4:     if $\mathscr{A}_i$ needs further decomposition then
5:         $\mathsf{ADD}(\mathscr{A}_i)$
```

To realise the $\mathsf{Decompose}(\mathscr{A})$ step in the ADD procedure, we apply our multiagent framework by letting each requirement $a \in \mathsf{R}$ act as a player and relevance $w(a,b)$ between to players a, b are determined by the relation between the corresponding requirements. A rational decomposition of the game is then the desired output of the process. For a thorough and more formal description of this agent-based model, the reader is referred to [9].

To demonstrate how our approach may be helpful to support software architecture design in real life, we provide three case studies below.

1.4.1 Case Study 1: SplitPay System

SplitPay is a mobile application based on Android platform. The purpose of this application is to manage shared expenses. Users in a group will post bills that they have paid for the group. Then, debts will be allocated to group users. The requirements of SplitPay are well documented in [12]. We elicit design elements from the documentation and organize them as three kinds of design driver: functional requirement, non-functional requirement and design constraints. Non-functional requirements in SplitPay's documentation are quite informal. We have to further analyze these non-functional requirements in order to correlate them with other design drivers. We use these elicited design drivers and their relationships as input to construct a decomposition game. Each design driver is a player in a decomposition game. Relationships between design drivers are weighted so that we are able to compose a evaluation function for every pair of design drivers. There are 18 functional requirements and 6 non-functional requirements. We compute a 10000-rational decomposition using PSA which converges in 6 iterations. The PSA-system has stablised on 5 coalitions. We show the results in Table 1.1. We only show the names of design drivers, where their exact meaning can be found in [12].

C_1	Performance1.1, Performance1.3, Function16.1, Function12.1, Function10.1
C_2	Function4.1,Function3.1,Function5.1,Performance1.2,Function15.1, Function14.1,Function11.1
C_3	Usability3.1,Function13.1,Function8.1,Function7.1,Function2.3, Function2.1,Function6.1,Function2.2,Function9.1,Function1.1
C_4	Performance1.4
C_5	Safety2.2

Table 1.1 Coalitions in SplitPay system. The coalition C_3 has the highest utility which highlights the importance of usability.

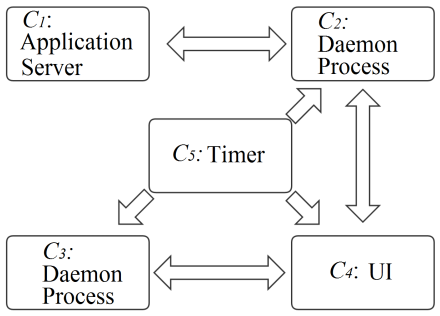

Fig. 1.2 SplitPay system architecture. Rectangular boxes represent system components and arrows represent their communications.

Based on this coalition structure, we derive a conceptual architecture as in Fig. 1.2. The decomposition highlights usability. User operations which require extra processing time will be handled in the daemon processes. Android systems can create service component for an application so that some processes will not affect user interface. C_2 can be a service component which listens from server and wraps up user requests to a formal HTTP request. C_3 is also a service component which updates a user's debt. C_1 is an application server which can install PHP and MySQL server.

1.4.2 Case Study 2: Wargame 2000

Wargame 2000 is a highly complex real-time ballistic missile defense simulation system. Non-functional requirements of the systems are given in [6]; the focus in [6] is to apply architecture tradeoff analysis method (ATAM) to identify sensitive points and risks in the design phase. Therefore all functional requirements, design constraints are omitted except non-functional requirements, which are well described. A scenario describes a non-functional requirement by stating the stimulus, the environmental conditions, and the measurable or observable response to the stimulus.

Since we are only able to access non-functional requirements from [6], our focus here is also on tradeoff analysis. Our decomposition game contains 12 refined scenarios under 7 general scenarios that include availability, reliability, performance, usability, modifiability, scalability and interoperability. We establish interactions between these scenarios. For example, consider the two scenarios:

- *Availability*: Simulation controller initiates execution (a game), starts subroutine processes, loads parameter files, and simulation starts within 10 minutes.
- *Performance*: System must perform all initialization activities within 10 minutes

These two scenarios are correlated because they both discuss the initialization procedure. We identify 48 such correlations between scenarios and compute a 10000-bounded decomposition, which is shown in Table 1.2.

C_1	Perfo3.1,Perfo3.2,Perfo3.3,Usabi4.1,Avala1.1,Avala1.2,Scala6.1,Scala6.2
C_2	Reliab2.1,Modif5.1,Modif5.2
C_3	Interop7.1

Table 1.2 Coalitions in Wargame 2000 system.

Similar to [6], we identify a conceptual software architecture design in Fig. 1.3. This decomposition reveals a high level architecture of Wargame2000, which is presented in Fig 1.2. Although this is a very coarse-grained architecture, it nevertheless reveals reasonable tradeoff among quality attributes. C_1 emphasises features for a simulator which requires real-time simulation and "human-in-loop" simulation. Some requierments (e.g. SCAL6.1) require a simulator to adapt new model. C_2 demands high quality of configuration of Wargame2000 system. C_3 builds a functionality to connect Wargame 2000 to other systems.

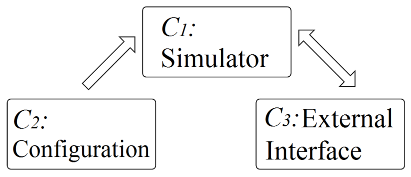

Fig. 1.3 Wargame system architecture.

1.4.3 Case Study 3: Cafeteria Ordering System

The aim of this case study is to apply our game model to a business management system. Cafeteria ordering system is a widely used case study for software design (see [14] for detailed description). The project is well documented in the literature. It consists of 11 non-functional requirements which belong to 6 types: availability (AVL), performance (PER), security (SEC), Usability (USE), robustness (ROB), safety (SAF). There are 49 functional requirements and a number of design constraints. We set bounds for a bounded decomposition to 5×10^5. A PSA system solves the game as shown in Table 1.3.

C_1	SI1.1,SI1.2
C_2	Pay.Method,Deliver.Location,Confirm.Response,Deliver.Select,Units.Multiple,Confirm.More
C_3	Menu.Available,Deliver.Times,Place.Register
C_4	USE1
C_5	AVL1,PER1,PER2,ROB1,SEC1,SEC2,USE2,SI1.3,SI2.1,SI2.2,SI2.4,CI1,CI2,UI2,UI3,SI2.3, SI2.5,Retrieve,Menu,Done.Patron,Deliver,Place,Done.Store,Pay,Done,Done.Cafeteria Done.Failure,Done.Inventory,Done.Menu,Confirm,Done.Time,Units
C_6	SEC4
C_7	Place.Date
C_8	Pay.Deliver,Pay.Deduct,Pay.Pickup
C_9	Menu.Date
C_{10}	SAF1
C_{11}	Confirm.Display
C_{12}	PER3,Place.Cutoff,Place.No,Deliver.Notimes,Units.TooMany,Pay.NG,Confirm.Prompt,Pay.OK

Table 1.3 Coalitions in COS.

A closer look reveals that it is reasonable to disregard some singleton coalitions. For example, USE1 in C_4 is from a business rule which only require the abidance of certain development standard; SAF1 in C_{10} is set up for indicating users' dietary precaution. These requirements are irrelevant to software architecture design. By removing these non-essential requirements, we have resolved a software architecture design in Fig. 1.4. This design is more detailed than the others. To work in an architectural level, requirements that we have elicited require more refinements.

For example, *Coalition 11* is just singleton coalition and it can be combined into *Coalition 5*; *Coalition 9* and *Coaltion 7* also can be combined into *Coalition 12*. As we can see, decomposition game does not only provide architecture design but also help designer to identify problems in requirements.

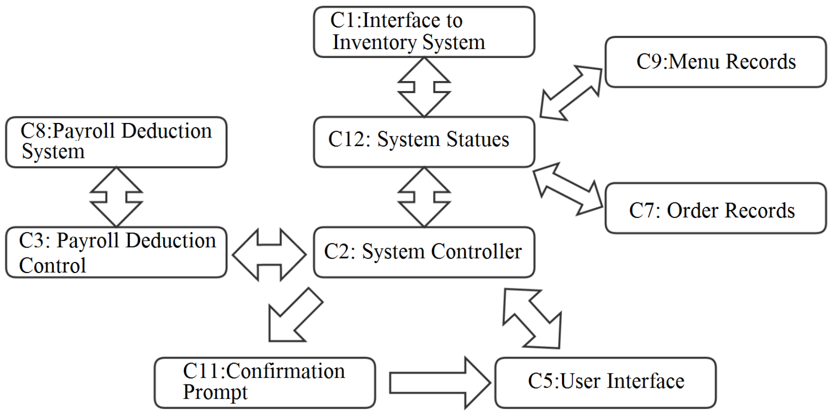

Fig. 1.4 COS architecture

1.5 Experiments

We conduct a number of experiments on synthetic games to evaluate the performance of our approach. Firstly, we calculate how much probability a bounded solution can be a core or a rational decomposition. Secondly, we measure the utilitarian welfare (sum of utilities) and the egalitarian welfare (the lowest utility) of the coalitions. Thirdly, we investigate the running times of our algorithm. We take the following parameters: a bound T for decompositions, the number n of players and a fixed value for α.

We fix a random model to generate decomposition games. The model firstly generates n players. Then, for each pair of players a and b, we randomly generate two weights within the range $[0,1]$. One weight is for the direction from a to b and the other is for the opposite direction. For each weight, we take a 50% probability to set it negative. We generate two types of games: *symmetric games* where the weights of the two directions (a,b) and (b,a) are the same, and *asymmetric games* where the two weights are not necessarily the same.

1. Results of Experiment 1 are shown in Fig. 1.5; here we investigate probability of a solution being in a core or a rational decomposition. In (a), we fix $n = 10$

players in the games while changing T from 200 to 2000. In (b), we also fix $n = 10$ players while changing α from 2 to 5 (note that this is different from our earlier description where α is determined by T, but rather is a fixed value). In (c), we fix $T = 2000$ while changing the number of players. The results show high probability of our approach finding a core or a rational decomposition. In addition, increasing T or α can both improve this probability. In particular, we notice in (c) that asymmetric games require much higher bounds for finding stable solutions if the number of players increases in decomposition games.

2. For Experiment 2, we use the same parameter setting as Experiment 1; the focus is to investigate utilitarian and egalitarian welfare obtained by decompositions games. The results of this experiment is shown in Fig. 1.6. The results in (a) and (b) change very little with varying parameters because a bounded solution has high probability in a core or a rational decomposition.
3. For Experiment 3, we follow the parameter setting from the previous two experiments; the focus here is to examine running times for finding a solution in decomposition games. The results of this experiment is shown in Fig. 1.7. With changing T in (a), the computation for each iteration is constance, and hence the time reflect the number of iterations before stability is reached. The running time shows a quadratic growth. With changing α in (b), the results are similar but running times increase exponentially. Thus we conclude that adopting the bound T to determine the values of α will lead to much faster computation while still achieving good accuracy.

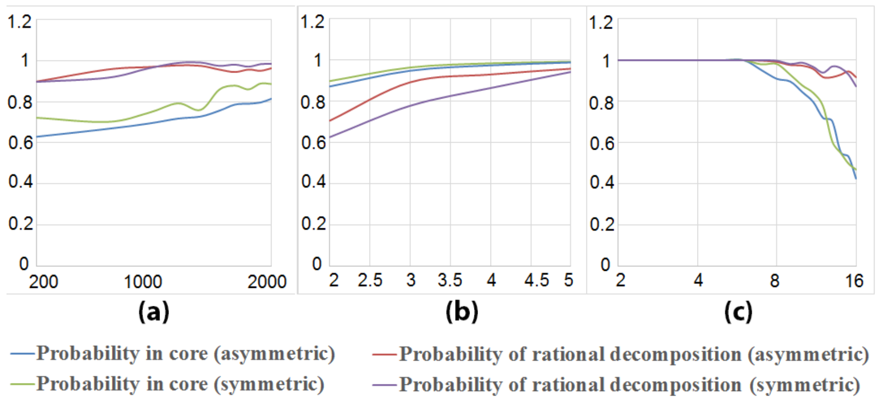

Fig. 1.5 Experiment 1: Comparison between probabilities of being in core and rational decomposition.

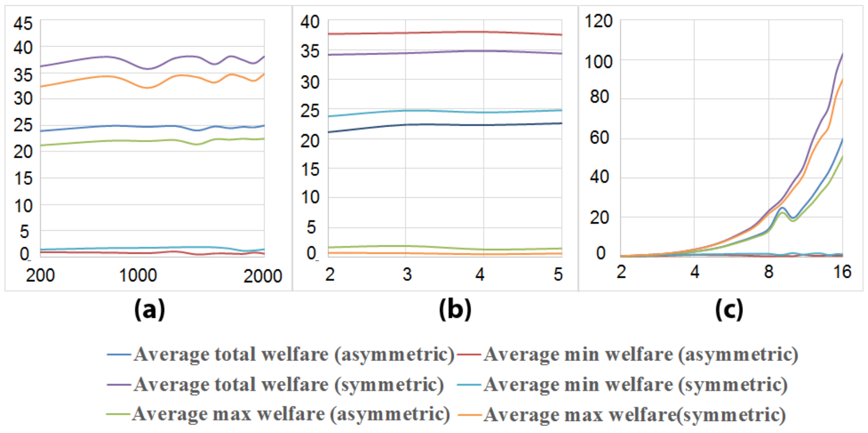

Fig. 1.6 Experiment 2: Achievements in utilitarian and egalitarian welfare.

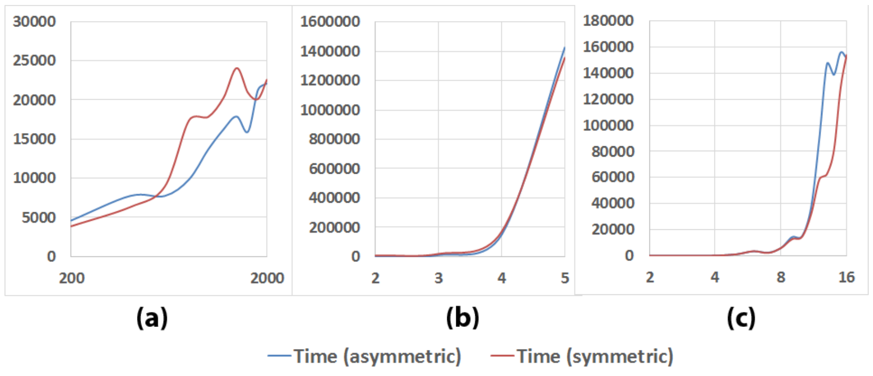

Fig. 1.7 Experiment 3: Time cost in changing T and α

1.6 Conclusion and Future Work

In this paper we simulate coalition formation based on decomposition games. The main difference between this work and our previous work [9] is that 1) here we use a multiagent framework that derive game solution in a distributed manner; 2) we put a bound on the complexity of query which improves efficiency. The case studies demonstrate that the approach can be used to support software architecture design by treating each requirement as a player. The experiments focus on performance of the solutions as well as time complexity and their results show that the multiagent framework can help us to achieve rational decompositions.

There are two directions for future works. Firstly, one may enrich the decomposition model by removing the assumption of altruistic agents and consider inhomogeneous individual payoffs. It would be interesting to investigate if common coalitional game solution concepts such as Shapley value give rise to meaningful software decompositions. Secondly, the current model assumes a waterfall model of software engineering process where the design process does not start until requirement analysis is finished. In other software engineering paradigms, requirements come and go in the software design phase and therefore we stipulate that the multiagent approach would help to dynamically determine software architectures.

References

1. Airiau, S. (2013): Cooperative games and multiagent systems. The Knowledge Engineering Review, 28(4), Cambridge University Press, 381–424.
2. Bai, Q., Liu, J., Wei, Z. (2015): Simulating and Modeling Dual Market Segmentation Using PSA Framework. In Proc of 2nd International Workshop on Smart Simulation and Modelling for Complex Systems.
3. Bass, L.(2007): Software architecture in practice. Pearson Education India.
4. Bass, L., Klein, M., Moreno, G. (2001): Applicability of general scenarios to the architecture tradeoff analysis method (No. CMU/SEI-2001-TR-014). Carnegie-Melon Univ., Soft. Eng. Inst.
5. Bass, L., Klein, M., and Bachmann, F.(2001): Quality attribute design primitives and the attribute driven design method. In Proc of PFE-4, Revised Papers from the 4th International Workshop on Software Product-Family Engineering. Springer, 169–186.
6. Jones, L., Lattanze, A. (2001): Using the architecture tradeoff analysis method to evaluate a wargame simulation system: A case study (No. CMU/SEI-2001-TN-022). Cargenie-Mellon Univ. Pittsburgh.
7. Liu, J., Wei, Z. (2014). From a Local to a Global Perspective of Community Detection in Networks. In PRICAI 2014, Trends in Artificial Intelligence (pp. 1036-1049). Springer International Publishing.
8. Liu, J., Wei, Z. (2014). Community Detection Based on Graph Dynamical Systems with Asynchronous Runs. In Computing and Networking (CANDAR), 2014 Second International Symposium on (pp. 463-469). IEEE.
9. Liu, J, Wei, Z. (2015): A game of attribute decomposition for software architecture design. To appear in Proc of the 12th International Colloquium on Theoretical Aspects of Computing (ICTAC 2015). Manuscript available at arxiv.org/abs/1508.02812
10. Shehory, O., Kraus, S. (1998): Methods for task allocation via agent coalition formation. Artificial Intelligence 101(1-2), 165–200.
11. Shoham, Y., Leyton-Brown, K. (2008): Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press.
12. SplitPay SRS. https://www.cise.ufl.edu/class/cen3031sp13/SRS_Example_1_2011.pdf
13. Tsvetovat, M., Sycara, K., Chen, Y., Ying, J. (2001): Customer coalitions in electronic markets, in: Agent-Mediated Electronic Commerce III, Springer, 121–138.
14. Wiegers, K., Beatty, J. (2013): Software requirements. Pearson Education.
15. Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., Wood, B. (2006): Attribute-Driven Design (ADD), Version 2.0 (No. CMU/SEI-2006-TR-023). Carnegie-Melon Univ., Soft. Eng. Inst.